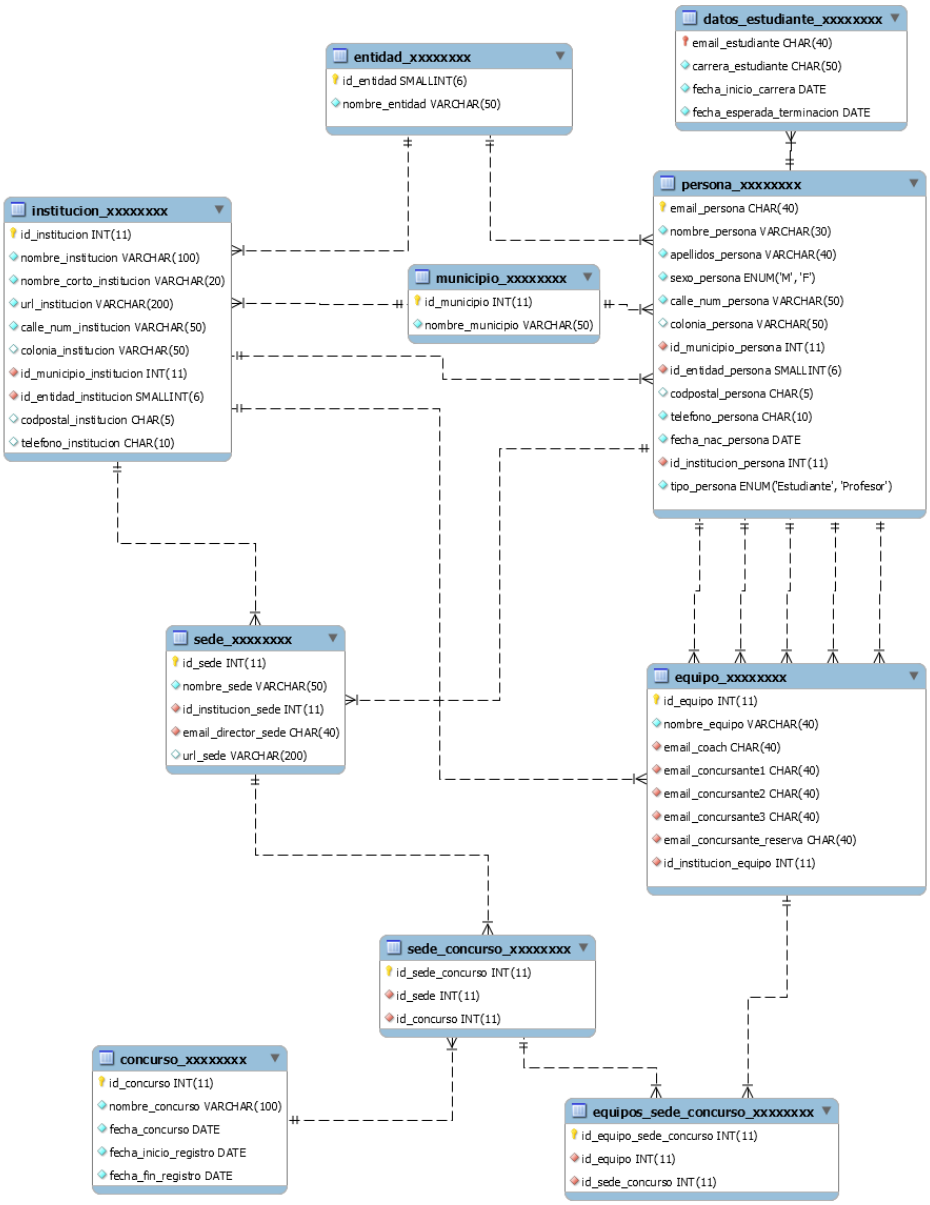
PROGRAMACION ORIENTADA A OBJETOS II

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 e institucion. El diagrama de la base de datos a utilizar se muestra a continuación (aunque las tablas no tendrán el sufijo _xxxxxx que representaban su matrícula en el nombre de las tablas en las prácticas 5, 6 y 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 `controlconcursos`), y colocar datos iniciales, se proporcionan los scripts creaconcursos.sql y llenaconcursos.sql

Con el propósito de probar sus DAOs, pueden escribir clientes de texto básicos donde creen objetos de alguna clase de entidad, por ejemplo, Institución, y a través del DAO general se llame a los métodos correspondientes para agregar, actualizar, obtener o modificar una Institución. 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, deberán colocarse en el paquete `poo2.progs.entidades`, y las implementaciones de los DAOs deberán colocare en el paquete `poo2.progs.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 `DAOConcursos`. 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 creaconcursos.sql y llenaconcursos.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 < creaconcursos.sql

mysql -u root -p  --default-character-set=utf8 < llenaconcursos.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 e institucion (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, aunque será asociado a otro repositorio para el Programa 1.

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 extendidas tienen como campos obligatorios los de la clase de la que heredan, y los constructores de las 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 nombreInstitucion en la clase Institución
    - El valor de nombreMunicipio en la clase Municipio
    - El valor de nombreEntidad en la clase Entidad
    - El valor de nombrePersona seguido de un espacio y del valor de apellidosPersona en la clase Persona
    - El valor de emailEstudiante en la clase DatosEstudiante
    - El valor de nombreConcurso en la clase Concurso
    - El valor de nombreEquipo en la clase Equipo
    - El valor de nombreSede en la clase Sede
    - El valor de nombreSede seguido de un espacio y el valor de nombreConcurso entre paréntesis, sin espacios entre los paréntesis y el contenido de los paréntesis para la clase SedeConcursoExtendida, por ejemplo: **`UAIE-UAZ (Gran Premio 2021)`**
    - El valor de nombreEquipo seguido del texto " en " (sin las comillas pero con el espacio antes y después de en), seguido del valor de nombreSede seguido de un espacio y el nombre del concurso entre paréntesis, sin espacios entre los paréntesis y el contenido de los paréntesis para la clase EquiposSedeConcursoExtendida, por ejemplo: **`Equipazo en UAIE-UAZ (Gran Premio 2021)`**
    - Las clases SedeConcurso y EquiposSedeConcurso no necesitan tener el método toString
    - Recuerde que la clase SedeConcursoExtendida hereda de la clase SedeConcurso agregándole los atributos nombreSede y nombreConcurso (ambos String) con sus respectivos getters y setters (los campos obligatorios serían los de la tabla sede_concurso) y que la clase EquposSedeConcursoExtendida hereda de la clase EquiposSedeConcurso agregándole los atributos nombreEquipo, nombreSede y nombreConcurso (todos String) con sus respectivos getters y setters (los campos obligatorios serían los de la tabla equipo_sede_concurso)

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

Se hace notar que las validaciones que serán hechas como parte final de la práctica 8 para el DAO de Institución, 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.
    - 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 es null, o un valor de 0 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 DaoPersona, se debe verificar que sexoPersona tenga como valor "M" o "F" y que tipoPersona tenga como valor "Estudiante" o "Profesor".
    - En el caso de DaoConcurso, el valor de fechaInicioRegistro debe ser previo al de fechaFinRegistro y el valor de fechaFinRegistro debe ser previo al valor de fechaConcurso.
    - En el caso de DaoDatosEstudiante, el valor de fechaInicioCarrera debe ser al menos 3 años previo al valor de fechaEsperadaTerminacion
    - En el caso de DaoSede, el valor de emailDirectorSede debe existir en la tabla persona como llave primaria y debe pertenecer a una persona de tipo "Profesor" que sea de la misma institución que la sede
    - En el caso de DatosPersona, el valor de emailEstudiante debe existir en la tabla persona como llave primaria y debe pertenecer a una persona de tipo "Estudiante"
    - En el caso de DaoEquipo, los valores de emailConcursante1, emailConcursante2, emailConcursante3 y emailConcursanteReserva (este último si es que no es null, pues es opcional) deben ser existir como llave primaria en la tabla persona y deben pertenecer a personas de tipo "Estudiante", el valor de emailCoach debe existir como llave primaria en la tabla persona y debe pertenecer a una persona de tipo "Profesor", todas los emails proporcionados para el equipo deben ser de personas de la misma institución y todos los emails deben ser diferentes, es decir, no puede usarse el mismo email al mismo tiempo para más de uno de los campos donde debe ir un email.
    - Si algún valor no es válido debería el método save o update 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, por ejemplo, una persona no se puede eliminar si ya está registrada en un equipo, una institución no se puede eliminar si ya está registrada en alguna persona, equipo o sede, etc. Si el registro a eliminar no existe o su valor de llave primaria ya aparece como valor en alguna otra tabla relacionada debe regresar false

Habiendo dejado claro esto, para cada tabla se debe implementar el DAO correspondiente (implementando la interface Dao<T>) y tales clases deberán llamarse DaoInstitucion, DaoPersona, DaoSede, DaoEntidad, DaoMunicipio, DaoConcurso, DaoEquipo, DaoDatosEstudiante, DaoSedeConcurso y DaoEquiposSedeConcurso. Todas estas clases deberán quedar en el paquete `poo2.progs.basedatos`

En las clases DaoEntidad y DaoMunicipio (que implementan la interface Dao<T> para las tablas entidad 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.

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.

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**.

NOTA: En el proyecto que clonan ya se encuentran todas las clases de entidad (incompletas), 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 las clases de manera manual

CON RESPECTO A LA CLASE DaoConcursosImpl

También deberá completarse el código de la clase DaoConcursosImpl que implementa la interface DaoConcursos (y que ya en la práctica 8 se dejará completo lo relacionado a la tabla institucion). La clase DaoConcursosImpl 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 DaoConcursosImpl. Esta clase también deberá quedar en el paquete `poo2.progs.basedatos`.

La interface DaoConcursosImpl, 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<Institucion> obtenInstituciones()** Este método devolverá una lista de instituciones en la tabla institución de la base de datos
- **public boolean agregaInstitucion(Institucion dato)** Este método agrega el usuario representado por el objeto u a la tabla Usuario (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 eliminaInstitucion(long idInstitucion)** Este método elimina la institución cuyo valor en el campo llave es igual al dado como argumento (idInstitucion) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). Este método debe devolver false en caso de que haya registros en la tabla sede, persona o equipos que hagan referencia a la institución a eliminar.
- **public boolean actualizaInstitucion(Institucion dato)** Este método modifica los datos de la institución representada por el objeto dato en la tabla institucion(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<Entidad> obtenEntidades()** Este método devolverá una lista de las entidades contenidos en la tabla entidad de la base de datos.

- **public List<Municipio> obtenMunicipios(long idEntidad)** Este método devolverá una lista de los municipios contenidos en la tabla municipio de la base de datos y que pertenezcan a la entidad cuyo ID es recibido como argumento

- **public List<String> obtenCorreosDeInstitucion(long idInstitucion, String tipo)** Este método devolverá la lista de correos de las personas de la institucion y tipo especificados por los argumentos (tomándolos de la tabla persona)

- **public List<Persona> obtenPersonas()** Este método devolverá una lista de Personas contenidos en la tabla persona de la base de datos
- **public boolean agregaPersona(Persona dato)** Este método agrega la persona representado por el objeto u a la tabla persona (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 eliminaPersona(String emailPersona)** Este método elimina la persona cuyo valor en el campo llave es igual al dado como argumento (emailPersona) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). Este método debe devolver false en caso de que haya registros en la tabla equipo, o sede que hagan referencia a la persona a eliminar..
- **public boolean actualizaPersona(Persona dato)** Este método modifica los datos de la persona representada por el objeto dato en la tabla persona(note que puede ser que no se pueda modificar el objeto por lo que regresa true si se pudo o false si no)

- **public Optional<DatosEstudiante> obtenDatosEstudiante(String emailEstudiante)** Este método devolverá la información del estudiante cuya llave primaria es igual al argumento recibido (emailEstudiante) representado como un objeto DatosEstudiante contenido dentro de un Optional, el Optional estará vacía si no hay ningún estudiante con el emailEstudiante indicado.
- **public boolean agregaDatosEstudiante(DatosEstudiante dato)** Este método agrega la película representada por el objeto p a la tabla datos_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 eliminaDatosEstudiante(String emailEstudiante)** Este método elimina los datos del estudiante cuyo email es recibido. Este método debe ya sea devolver false en caso de que no se pueda eliminar.
- **public boolean actualizaDatosEstudiante(DatosEstudiante dato)** Este método modifica los datos del estudiante representado por el objeto dato en la tabla datos_estudiante (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<Sede> obtenSedes()** Este método devolverá una lista con los registros de la tabla sede.
- **public boolean eliminaSede(long idSede)** Este método elimina el registro de la tabla sede cuyo valor de llave primaria es el recibido como argumento (note que puede ser que no se puedan eliminar por lo que regresa true si se pudo o false si no). Este método debe devolver false en caso de que haya registros en la tabla sede_concurso o equipos_sede_concurso hagan referencia a la sede a eliminar.
- **public boolean agregaSede(Sede dato)** Este método inserta un registro en la tabla sede con los datos recibidos en el argumento (note que puede ser que no se pueda agregar por lo que regresa true si se pudo o false si no)
- **public boolean actualizaSede(Sede dato)** Este método actualiza un registro en la tabla sede con los datos recibidos en el argumento (note que puede ser que no se pueda agregar por lo que regresa true si se pudo o false si no)

- **public List<Concurso> obtenConcursos()** Este método devolverá los registros de la tabla concurso.
- **public boolean eliminaConcurso(long idConcurso)** Este método elimina el concurso un valor en el campo de llave primaria sea igual al recibido como argumento (note que puede ser que no se puedan eliminar por lo que regresa true si se pudo o false si no). Este método debe devolver false en caso de que haya registros en la tabla sede_concurso o equipos_sede_concurso que hagan referencia al concurso a eliminar.
- **public boolean agregaConcurso(Concurso dato)** Este método inserta un registro en la tabla concurso con los datos recibidos (note que puede ser que no se pueda agregar por lo que regresa true si se pudo o false si no)
- **public boolean actualizaConcurso(Concurso dato)** Este método actualiza un registro en la tabla concurso con los datos recibidos (note que puede ser que no se pueda agregar por lo que regresa true si se pudo o false si no)

- **public List<Equipo> obtenEquipos()** Este método devolverá los registros de la tabla equipo.
- **public boolean eliminaEquipo(long idEquipo)** Este método elimina el equipo un valor en el campo de llave primaria sea igual al recibido como argumento (note que puede ser que no se puedan eliminar por lo que regresa true si se pudo o false si no). Este método debe devolver false en caso de que haya registros en la tabla equipos_sede_concurso que hagan referencia al equipo a eliminar.
- **public boolean agregaEquipo(Equipo dato)** Este método inserta un registro en la tabla equipo con los datos recibidos (note que puede ser que no se pueda agregar por lo que regresa true si se pudo o false si no)
- **public boolean actualizaEquipo(Equipo dato)** Este método actualiza un registro en la tabla equipo con los datos recibidos (note que puede ser que no se pueda agregar por lo que regresa true si se pudo o false si no)

- **public List<Sede> obtenSedesDisponibles(long idConcurso)** Este método devolverá una lista con las sedes disponibles para el concurso especificado como argumento, entendiendo como disponibles aquellas sedes que no han sido aún registradas para el concurso indicado.
- **public List<SedeConcursoExtendida> obtenSedesAsignadas(long idConcurso)** Este método devolverá una lista con los sedes_concurso ya asignadas al concurso especificado como argumento, es decir, que tienen un registro en la tabla sede_concurso con el campo id_concurso igual al argumento recibido
- **public boolean agregaSedeConcurso(SedeConcurso dato)** Este método agrega un registro en la tabla sede_concurso con la información recibida en el objeto dato, (note que puede ser que no se puedan modificar por lo que regresa true si se pudo o false si no)
- **public boolean eliminaSedeConcurso(long idSedeConcurso)** Este método elimina registro de la tabla sede_concurso cuyo valor de llave primaria concuerda con el valor recibido (note que puede ser que no se puedan modificar por lo que regresa true si se pudo o false si no). Este método debe devolver false en caso de que haya registros en la tabla equipos_sede_concurso que hagan referencia a la sede_concurso a eliminar.

- **public List<EquiposSedeConcursoExtendida> obtenEquiposRegistrados(long idConcurso, long idInstitucion)** Este método devolverá una lista de objetos EquiposSedeConcursoExtendida con la información de los equipos de la institución indicada por el segundo argumento que ya se encuentren registrados al concurso especificado como primer argumento, esto implica tomar registros de la tabla equipos_sede_concurso que pertenezcan al equipo indicado y que tengan

como id_sede_concurso el valor de algún registro de la tabla sede_concurso asociado con el concurso indicado.

-
- **public List<Equipo> obtenEquiposDisponibles(long idSedeConcurso, long idInstitucion)** Este método devolverá una lista con los equipos disponibles para la sede_concurso especificado como primer argumento y que sean de la institución indicada en el segundo argumento, entendiendo como disponibles aquellos equipos que no han sido aún registradas para el concurso asociado al valor de sede_concurso.

- **public boolean public boolean registrarEquipoSedeConcurso(EquiposSedeConcurso dato)** Este método agrega un registro en la tabla equipos_sede_concurso con la información recibida en el objeto dato, (note que puede ser que no se puedan modificar por lo que regresa true si se pudo o false si no)
- **public boolean cancelarEquipoSedeConcurso(long idEquipoSedeConcurso)** Este método elimina un registro de la tabla equipos_sede_concurso cuyo valor de llave primaria concuerda con el valor recibido (note que puede ser que no se puedan modificar 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 la clase 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 la clase de entidad Institucion: (3.5%)

CLASES DAO Individuales (64%):

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

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

CLASE DAOConcursosImpl: 30%

- Métodos que tienen que ver con Institucion: 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 Institucion, Persona, Sede, Entidad, Municipio, Concurso, Equipo y DatosEstudiante: (0.5% 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 SedeConcurso y EquiposSedeConcurso: (0.4% cada clase que cumpla con todo lo indicado)

- Agregado de la implementación de Serializable, los 3 constructores, los getters y setters para los campos extra a los heredados y el método toString a las clases de entidad SedeConcursoExtendida y EquiposSedeConcursoExtendida: (0.6% cada clase que cumpla con todo lo indicado)

CLASES DAO Individuales (64%):

- Implementación de Dao<Institucion> en clase DaoInstitucion : 2%
- Implementación de Dao<Entidad> en clase DaoEntidad: 3%
- Implementación de Dao<Municipio> en clase DaoMunicipio: 3%
- Implementación de Dao<Persona> en clase DaoPersona: 12%
- Implementación de Dao<DatosEstudiante> en clase DaoDatosEstudiante: 8%
- Implementación de Dao<Sede> en clase DaoSede: 9%
- Implementación de Dao<Concurso> en clase DaoConcurso: 7%
- Implementación de Dao<Equipo> en clase DaoEquipo: 12%
- Implementación de Dao<SedeConcurso> en clase DaoSedeConcurso: 4%
- Implementación de Dao<EquiposSedeConcurso> en clase DaoEquiposSedeConcurso: 4%

CLASE DAOConcursoImpl: 30%

- Métodos que tienen que ver con Institucion: 0.25%
- Métodos que tienen que ver con Entidad: 0.25%
- Métodos que tienen que ver con Persona: 0.5%
- Métodos que tienen que ver con DatosEstudiante: 0.5%
- Métodos que tienen que ver con Sede: 0.5%
- Métodos que tienen que ver con Concurso: 0.5%
- Métodos que tienen que ver con Equipo: 0.5%
- Métodos que tienen que ver con SedeConcurso: 0.5%
- Métodos que tienen que ver con EquiposSedeConcurso: 0.5%
- Método obtenMunicipios: 2%
- Método obtenCorreosDeInstitucion: 2%
- Método obtenSedesDisponibles: 5%
- Método obtenSedesAsignadas: 5%
- Método obtenEquiposRegistrados: 5%
- Método obtenEquiposDisponibles: 7%

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 10 de octubre 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**.