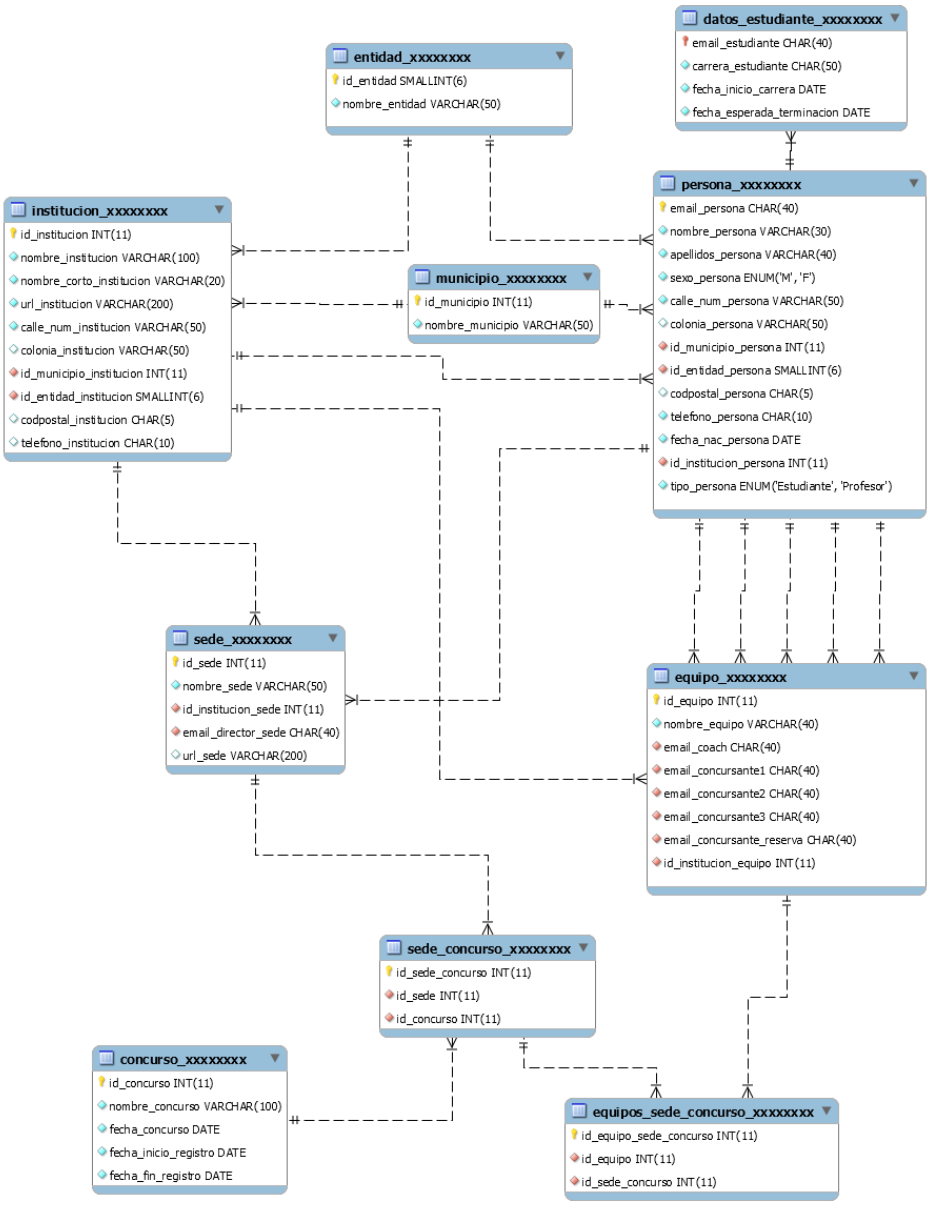
PROGRAMACION ORIENTADA A OBJETOS II

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 realizó en la práctica 8 donde se desarrolló el DAO para la tabla institución. 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 `mx.edu.uaz.ingsoftware.poo2.entidade`s, 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 `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 `mx.edu.uaz.ingsoftware.poo2.interfaces`

INSTRUCCIONES PARA INICIALIZAR EL PROYECTO DEL PROGRAMA 1

ES IMPORTANTE QUE LAS SIGUIENTES INSTRUCCIONES LAS SIGA EN EL ORDEN INDICADO:

1. Copie el proyecto IntellijIDEA de la Práctica 08 haciendo lo siguiente:
    a. Teniendo abierto el proyecto de la Práctica 08, dé click con el botón derecho en el ícono de Gradle que se encuentra en el borde derecho, expanda Tasks -> build y dé doble click en la opción Clean
    b. Dé click con el botón derecho en el nombre del proyecto (en la sección Projects que se encuentra en la parte izquierda) y seleccione la opción Copy -> Copy
    c. Dé click con el botón derecho en el nombre del proyecto (en la sección Projects que se encuentra en la parte izquierda) y seleccione Paste, en el cuadro de diálogo que aparece ponga como nombre del proyecto **Programa 1 POO2** asegurándose que en el campo Directory está su directorio de proyectos de IntellijIDEA (con el formato C:\Users*NomUsuario*\IdeaProjects en Windows donde *NomUsuario* es su nombre de usuario en Windows) y dé click en OK
    d. Cierre el proyecto actual (que es el de la Práctica 08)
2. Abra el proyecto nuevo y haga modificaciones a su contenido:
    a. Dé click en la opción Open or Import para abrir el nuevo proyecto
    b. Seleccione en el cuadro de diálogo la ruta del proyecto que acaba de copiar en los pasos anteriores
    c. Si al abrirlo le marca el error Invalid VCS Root Mapping , dé click en Configure, seleccione el directorio de la lista que aparece, dé click en el − que está a la derecha para eliminarlo, y dé click en OK
    d. En la vista Project Files (en la extrema izquierda superior), expanda la carpeta del proyecto y elimine la carpeta .github y el archivo README.md
    e. En la vista Project (en la extrema izquierda superior), expanda el proyecto y edite el archivo settings.gradle para que el nombre del proyecto sea **Programa 1 POO2**
    f. Edite el archivo .gitignore dentro de la carpeta .idea para que la ruta al DataSources sea la correcta (cambiar la parte donde dice Practica 08 Lab POO2 por **Programa 1 POO2**)

3. Habilite la integración con Git, combinando lo que tiene el repositorio remoto con el código que copió de la Práctica 8:
    a. Dé click en la opción Version Control Integration del menu VCS, seleccione Git y de click en OK
    b. Entre al menú VCS -> Git -> Remotes..., de click en el + para agregar el URL del repositorio remoto (que es el proporcionado al aceptar la tarea), dar click en Ok
    c. Entre al menú VCS -> Git -> Fetch
    d. Entre al menú VCS -> Git -> Pull seleccionando el branch master y dé click en el botón Pull
    e. Haga un primer Commit con el mensaje "Inicializacion de Codigo para Programa 1" incluyendo solo los archivos de las carpeta src, .idea **(de este desmarque el archivo .gitignore)** , y gradle\wrapper además de los archivos de la carpeta raiz del proyecto (dar click en el icono Group By, el que tienen cuatro cuadritos, y seleccionar By Directory para que aparezcan por directorio y pueda seleccionar tales carpetas y archivos más fácilmente)
    f. Entre al menu VCS -> Git -> Push... y en el cuadro de dialogo que aparece de click en el botón Push

4. Ya puede comenzar a completar la implementación necesaria para el Programa 1 de acuerdo a las indicaciones de las siguientes secciones.

NOTA IMPORTANTE: Durante 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
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)
- El método equals y el método hashCode usando solamente el campo que es llave primaria
- El método String 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
    - El valor de emailPersona 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, por ejemplo: **`UAIE-UAZ (Gran Premio 2020)`**
    - 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, por ejemplo: **`Equipazo en UAIE-UAZ (Gran Premio 2020)`**
    - 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_ede_concurso)

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

Se hace notar que las validaciones que fueron 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 si tiene 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 asigno al campo autoncrementable lo pueden encontrar en el código de las prácticas 6 y 7).
  - Si algún valor no es válido debería el método save o update regresar false.
- Al eliminar es necesario determinar 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 `mx.edu.uaz.ingsoftware.poo2.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.

Un cambio con respecto a lo que se realizó en la Práctica 8 es que no comunicaremos los errores que se pudieran presentar al hacer alguna operación en la base de datos (**a excepción de los constructores que 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 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 DaoConcursosImpl

También deberá completarse el código de la clase DaoConcursosImpl que implementa la interface DaoConcursos (y que ya en la práctica 8 se dejó completo lo relacionado a la tabla estudiante). 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 `mx.edu.uaz.ingsoftware.poo2.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 DatosEstudiante obtenDatosEstudiante(String emailEstudiante)** Este método devolverá la información del estudiante cuya llave primaria es igual al argumento recibido (emailEstudiante)
- **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<SedeConcursoExtendida> obtenSedesDisponibles(long idConcurso)** Este método devolverá una lista con los sedes_concurso 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 idSedeConcurso, long idInstitucion)** Este método devolverá una lista con los equipos de la institución indicada ya registrados al concurso especificado como argumento, es decir, que tienen un registro en la tabla equipo_sede_concurso con el campo id_sede_concurso igual al primer argumento recibido donde el correspondiente equipo pertenece a la institución con id dado como segundo argumento.
- **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 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:

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: 9%

- Implementación de Dao<Equipo> en clase DaoEquipo: 10%

- 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: 3%
- Método obtenCorreosDeInstitucion: 3%
- Método obtenSedesDisponibles: 6%
- Método obtenSedesAsignadas: 4%
- Método obtenEquiposRegistrados: 4%
- Método obtenEquiposDisponibles: 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

SE HACE MENCION DE QUE DADO QUE EN EL EVALUADOR DE GITHUB SOLO SE PERMITEN CALIFICACIONES ENTERAS, ALGUNAS PRUEBAS EN EL REPOSITORIO REMOTO SE REALIZARAN AGRUPADAS PARA QUE RESULTE EN PUNTOS ENTEROS, ASÍ, POR EJEMPLO, EN UN GRUPO SE PONDRÁ LA CLASE DE ENTIDAD Institucion Y LA CLASE DE ENTIDAD Entidad, Y SI EN ESE GRUPO AMBAS CLASES DE ENTIDAD CUMPLEN CON LOS REQUISITOS, SE SUMARA 1 PUNTO, SI ALGUNA NO CUMPLE, NO SE DARAN PUNTOS POR ESE GRUPO.

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 25 de octubre del 2020. 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**.