PROGRAMACION ORIENTADA A OBJETOS II
PROGRAMA 2

Para este programa, deberá completar los servicios REST que se comenzarán a desarrollar para la práctica 9, los cuales deben poder ser usados por alguna implementación de la interface DaoConcursos del Programa 1, es decir, los servicios REST a generar, deben proveer la información requerida **SOLAMENTE** por los métodos indicados en la interface DAOConcursos (no es necesario implementar todos los métodos de los DAOs individuales del Programa 1).

Entonces, tomando en cuenta de la práctica que el URL Base de los servicios a proporcionar es http://localhost:8080/RESTConcursos-XXXXXXXX/servicios (donde XXXXXXXX se sustituye por su matrícula, el programa 2 deberá contener un servicio REST por cada una de las tablas (entidad, municipio, institucion, persona, datos_estudiante, sede, concurso, equipo, sede_concurso y equipo_sede_concurso) que componen a la base de datos controlconcursos, cada uno representado por la ruta indicada más adelante.

INSTRUCCIONES PARA INICIALIZAR EL PROYECTO DEL PROGRAMA 2 (Y PRÁCTICA 9)

Paso 1. 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 2. Con la guía del instructor, se estará demostrando lo que hay que realizar para el Programa 2, completando específicamente lo relacionado a los servicios REST para las tablas municipio y estudiante (la calificación de la Práctica 9 es en base a los servicios de estas dos tablas).

Paso 3. Una vez completado lo relacionado a la Práctica 9, podrá continuar de manera independiente lo restante. Para ello se deberá copiar el proyecto que tenga la Práctica 9 ya completada en otro proyecto que se asociará con el repositorio del Programa 2. Pueden encontrar mayores detalles de cómo hacer esto en el archivo README.md

NOTA IMPORTANTE: Las pruebas del programa 2 estan empacadas en un archivo ZIP para que puedan compilar las de la Práctica 9 sin necesidad de tener todo lo del Programa 2. Una vez que se tengan todas las clases de entidad completas ya será seguro desempacar el archivo ZIP. Las pruebas del programa 2 están en el paquete poo2.progs.main dentro de la sección test. Ver archivo README.md para mayores detalles.

MODIFICACIÓN DE LOS ARCHIVOS PROPORCIONADOS (EN CUANTO A CONFIGURACIÓN)

Una vez que tengas el repositorio local, el trabajo principal consiste en crear las clases para proporcionar los servicios REST descritos en la siguiente sección. Aseguráte de una vez creados los servicios REST, poner en el método agregaRecursosREST de la clase AplicacionConfig código para agregar las clases que representan a los servicios REST al Set que recibe como argumento.

Antes de hacer los cambios en el código haga los pasos indicados en la sección CONFIGURACION INICIAL del archivo README.md

TRABAJO A REALIZAR EN CUANTO A CÓDIGO

De acuerdo a lo indicado para el Programa 1, la interface ***DaoConcursos*** contiene los siguientes métodos:

1. **public List<Carrera> obtenInstituciones()**
2. **public boolean agregaInstitucion(Institucion dato)**
3. **public boolean eliminaInstitucion(long idInstitucion)**
4. **public boolean actualizaInstitucion(Institucion dato)**

5. **public List<Entidad> obtenEntidades()**

6. **public List<Municipio> obtenMunicipios(long idEntidad)**

7. **public List<Persona> obtenPersonas()**
8. **public boolean agregaPersona(Persona dato)**
9. **public boolean eliminaPersona(String emailPersona)**
10. **public boolean actualizaPersona(Persona p)**
11. **public List<String> obtenCorreosDeInstitucion(long idInstitucion, String tipo)**

Los métodos en verde son métodos que deben no corresponde a los 5 métodos que ya pueden realizar los servicios REST por heredar de RESTAbstracto

12. **public DatosEstudiante obtenDatosEstudiante(String emailEstudiante)**
13. **public boolean agregaDatosEstudiante (DatosEstudiante dato)**
14. **public boolean eliminaDatosEstudiante (String emailPersona)**
15. **public boolean actualizaDatosEstudiante (DatosEstudiante dato)**

16. **public List<Sede> obtenSedes()**
17. **public boolean agregaSede(Sede dato)**
18. **public boolean eliminaSede(long idSede)**
19. **public boolean actualizaSede(Sede dato)**
20. **public List<Sede> obtenSedesDisponibles(long idConcurso)**

21. **public List<Concurso> obtenConcursos()**
22. **public boolean agregaConcurso(Concurso dato)**
23. **public boolean eliminaConcurso(long idConcurso)**
24. **public boolean actualizaConcurso(Concurso dato)**

25. **public List<Equipo> obtenEquipos()**
26. **public boolean agregaEquipo(Equipo dato)**
27. **public boolean eliminaEquipo(long idEquipo)**
28. **public boolean actualizaEquipo(Equipo dato)**
29. **public List<Equipo> obtenEquiposDisponibles(long idSedeConcurso, long idInstitucion)**

30. **public List<SedeConcursoExtendida> obtenSedesAsignadas(long idConcurso)**
31. **public boolean agregaSedeConcurso(SedeConcurso dato)**
32. **public boolean eliminaSedeConcurso(long idSedeConcurso)**

33. **public List<EquiposSedeConcursoExtendida> obtenEquiposRegistrados(long idConcurso, long idInstitucion)**
34. **public boolean registrarEquipoSedeConcurso(EquiposSedeConcurso dato)**
35. **public boolean cancelarEquipoSedeConcurso (long idEquipoSedeConcurso)**

Para todos estos métodos debe crear servicios REST que realicen el trabajo correspondiente.

Los métodos 1 a 4 deberían estar en un servicio REST de ruta **institucion**. El método 5 debería estar en un servicio REST de ruta **entidad**. El método 6 debería estar en un servicio REST de ruta **municipio**. Los métodos 7 a 11 deberían estar en un servicio REST de ruta **persona**. Los métodos 12 a 15 deberían estar en un servicio REST de ruta **datosestudiante**. Los métodos 16 a 20 deberían estar en un servicio REST de ruta **sede**. Los métodos 21 a 24 deberían estar en un servicio REST de ruta **concurso**. Los métodos 25 a 29 deberían estar en un servicio REST de ruta **equipo**. Los métodos 30 a 32 deberían estar en un servicio REST de ruta **sedeconcurso**. Los métodos 33 a 35 deberían estar en un servicio REST de ruta **equipossedeconcurso**.

Las clases de entidad a generar deberán estar en el paquete poo2.progs.entidades y los servicios en el paquete poo2.progs.servicios

Algunos de estos métodos ya tienen prácticamente la funcionalidad deseada, puesto que los servicios REST heredan de la clase RESTAbstracto, mientras que otros requieren de la generación de código más elaborado para realizar el trabajo que se requiere. La ruta específica que debe tener el servicio REST asociado con los métodos de la interface DAOConcursos son los siguientes (urlbase es el URL Base de los servicios REST, indicado ya previamente):

Método de la Interface DAOConcursos	**Ruta del Servicio REST**	**Método HTTP a usar**
obtenInstituciones	urlbase/institucion	GET
agregaInstitucion	urlbase/institucion	POST
actualizaInstitucion	urlbase/institucion/{idinst}	PUT
eliminaInstitucion	urlbase/institucion/{idinst}	DELETE
obtenEntidades	urlbase/entidad	GET
obtenMunicipios	urlbase/municipio/{identidad}	GET
obtenPersonas	urlbase/persona	GET
agregaPersona	urlbase/persona	POST
actualizaPersona	urlbase/persona/{emailpersona}	PUT
eliminaPersona	urlbase/persona/{emailpersona}	DELETE
obtenCorreosDeInstitucion	urlbase/persona/correos/{idinst}/{tipopersona}	GET
obtenDatosEstudiante	urlbase/datosestudiante	GET
agregaDatosEstudiante	urlbase/datosestudiante	POST
actualizaDatosEstudiante	urlbase/datosestudiante/{email}	PUT
eliminaDatosEstudiante	urlbase/datosestudiante/{email}	DELETE
obtenSedes	urlbase/sede	GET

agregaSede	urlbase/sede	POST
actualizaSede	urlbase/sede/{idsede}	PUT
eliminaSede	urlbase/sede/{idsede}	DELETE
obtenSedesDisponibles	urlbase/sede/disponibles/{idconc}	GET
obtenConcursos	urlbase/concurso	GET
agregaConcurso	urlbase/concurso	POST
actualizaConcurso	urlbase/concurso/{idconcurso}	PUT
eliminaConcurso	urlbase/concurso/{idconcurso}	DELETE
obtenEquipos	urlbase/equipo	GET
agregaEquipo	urlbase/equipo	POST
actualizaEquipo	urlbase/equipo/{idequipo}	PUT
eliminaEquipo	urlbase/equipo/{idequipo}	DELETE
obtenEquiposDisponibles	urlbase/equipo/disponibles/ {idsedeconcurso}/{idinst}	GET
obtenSedesAsignadas	urlbase/sedeconcurso/asignadas/ {idconcurso}	GET
agregaSedeConcurso	urlbase/sedeconcurso	POST
eliminaSedeConcurso	urlbase/sedeconcurso/{idsedeconc}	DELETE
obtenEquiposRegistrados	urlbase/equipossedeconcurso/ registrados/{idconc}/{idinst}	GET
registrarEquipoSedeConcurso	urlbase/equipossedeconcurso	POST
cancelarEquipoSedeConcurso	urlbase/equipossedeconcurso/ {idequiposedeconc}	DELETE

Para que los servicios REST funcionen correctamente, es necesario que las clases de entidad correspondiente tengan las anotaciones necesarias, para la práctica 9 se estará agregando las anotaciones a algunas de ellas, las cuales pueden usar de referencia para completar las restantes. Para las clases extendidas (SedeConcursoExtendida y EquiposSedeConcursoExtendida solo es necesario poner la anotación @XmlRootElement a la clase (dado que esas no representan a una tabla de la base de datos).

Al igual que para los servicios a realizarse en la práctica 9, las operaciones para agregar, actualizar o eliminar deberán primero validar si es posible realizar la operación solicitada, de acuerdo a las reglas indicadas en el Programa 1, si no es posible, regresará "false".

NOTA: El servicio REST asociado con el método obtenCorreosDeInstitucion deberá regresar un String compuesto por todos los correos que sean de la institución y del tipo de persona indicados, separando a cada correo con un punto y coma.

ESPECIFICACIONES IMPORTANTES QUE DEBEN CUMPLIR LAS CLASES DE ENTIDAD

1. Las clases de entidad deben tener las siguientes anotaciones:
    a. `@Entity` que indica que esta clase representa una tabla en alguna base de datos
    b. `@Table(name="nombretabla")` que indica explícitamente el nombre de la tabla que representa la clase

    c. `@XmlRootElement` que representa que la información de esta clase puede ser convertida a un formato XML (uno de los dos comúnmente usados en la transferencia de información entre cliente y servicio REST)
2. Los atributos de la clases de entidad deberán tener anotaciones que especifican información asociada con la columna correspondiente en la tabla:
    a. `@Column(name = "nombrecolumna")` para indicar a que columna está relacionado el atributo
    b. @Id si es que el atributo está relacionado a la llave primaria de la tabla
    c. `@Basic(optional = false)` si es que el campo en la tabla es obligatorio
    d. `@NotNull` en caso de que la columna correspondiente tenga el atributo NOT NULL y sea representado en Java con una clase (no con un tipo primitivo)
    e. Se pueden usar anotaciones para solicitar que JPA haga ciertas validaciones, por ejemplo, una longitud mínimo y/o una máxima usando `@Size(min = LONGMIN, max = LONGMAX)`, no es necesario poner min y max, se ponen solo los deseados
    f. Se puede especificar que cumpla con alguna expresión regular usando @Pattern(regexp =EXPRESIONREGULAR, message=MENSAJEERROR), donde EXPRESIONREGULAR es la expresión regular que se requiere cumpla el valor a guardar y message contiene el mensaje a regresar en la excepción que se genera si el campo no cumple con la expresión regular

ESPECIFICACIONES IMPORTANTES QUE DEBEN CUMPLIR LOS SERVICIOS REST

1. Las clases que representan a los servicios REST deben tener un constructor vacío que manda llamar al constructor de la clase padre pasándole como referencia la clase de entidad con la que se trabajará.
2. Deben tener un atributo privado de tipo **EntityManager** que tendrá la anotación siguiente: **`@PersistenceContext(unitName = "default")`**
3. El método **getEntityManager** (que debe implementar) regresará simplemente el objeto **EntityManager** que tiene como atributo
4. Solo implementará los métodos que requiera hacer para la tabla para la cual está haciendo los servicios REST con las anotaciones adecuadas (GET, POST, PUT, DELETE, si consume información de que tipo, si genera información de que tipo y las rutas asociadas)

CON RESPECTO A LAS PRUEBAS EN GITHUB

Para este programa y práctica, las pruebas que se realizarán en Github requerirán que el servidor Glassfish este corriendo en su computadora y el servidor MySQL que se estará usando será el de su computadora. Para que estas pruebas puedan por tanto funcionar, se requiere de instalar un software que permita a Github comunicarse con su servidor Glassfish y servidor MySQL de manera remota. El software que usaremos será ngrok y en las sesiones finales donde se estará trabajando en la práctica 9, se demostrará como configurar ngrok y como ejecutarlo antes de hacer un push al repositorio remoto (que tenga como fin ver cómo funcionan las pruebas en Github). Esto implicará estar modificando un archivo de configuración (datosmysql.properties) antes de hacer cada push donde ya se quiera estar haciendo pruebas en el repositorio remoto. Mas detalles en el archivo README.md

PORCENTAJE DE CALIFICACION DE CADA ELEMENTO PARA PRACTICA 9:

La calificación de cada servicio es el siguiente:

Método de la Interface DAOEscolares	Ruta del Servicio REST	Puntos
obtenMunicipios	urlbase/municipio/{identidad}	20
obtenInstituciones	urlbase/institucion	8
agregaInstitucion	urlbase/institucion	24
actualizaInstitucion	urlbase/institucion/{idinst}	24
eliminaInstitucion	urlbase/institucion/{idinst}	24

PORCENTAJE DE CALIFICACION DE CADA ELEMENTO PARA PROGRAMA 2:

La calificación de cada servicio es la siguiente:

Método de la Interface DAOConcursos	Ruta del Servicio REST	Puntos
obtenInstituciones	urlbase/institucion	1
agregaInstitucion	urlbase/institucion	1
actualizaInstitucion	urlbase/institucion/{idinst}	1
eliminaInstitucion	urlbase/institucion/{idinst}	1
obtenEntidades	urlbase/entidad	3
obtenMunicipios	urlbase/municipio/{identidad}	1
obtenPersonas	urlbase/persona	3
agregaPersona	urlbase/persona	5
actualizaPersona	urlbase/persona/{emailpersona}	5
eliminaPersona	urlbase/persona/{emailpersona}	5
obtenCorreosDeInstitucion	urlbase/persona/correos/{idinst}/{tipopersona}	4
obtenDatosEstudiante	urlbase/datosestudiante/{email}	3
agregaDatosEstudiante	urlbase/datosestudiante	3
actualizaDatosEstudiante	urlbase/datosestudiante/{email}	3
eliminaDatosEstudiante	urlbase/datosestudiante/{email}	2
obtenSedes	urlbase/sede	3
agregaSede	urlbase/sede	4
actualizaSede	urlbase/sede/{idsede}	4
eliminaSede	urlbase/sede/{idsede}	4
obtenSedesDisponibles	urlbase/sede/disponibles/{idconc}	2.5
obtenConcursos	urlbase/concurso	3
agregaConcurso	urlbase/concurso	4
actualizaConcurso	urlbase/concurso/{idconcurso}	4

eliminaConcurso	urlbase/concurso/{idconcurso}	2
obtenEquipos	urlbase/equipo	3
agregaEquipo	urlbase/equipo	6
actualizaEquipo	urlbase/equipo/{idequipo}	6
eliminaEquipo	urlbase/equipo/{idequipo}	4
obtenEquiposDisponibles	urlbase/equipo/disponibles/ {idsedeconcurso}/{idinst}	2.5
obtenSedesAsignadas	urlbase/sedeconcurso/asignadas/ {idconcurso}	2.5
agregaSedeConcurso	urlbase/sedeconcurso	3
eliminaSedeConcurso	urlbase/sedeconcurso/{idsedeconc}	3
obtenEquiposRegistrados	urlbase/equipossedeconcurso/ registrados/{idconc}/{idinst}	2.5
registrarEquipoSedeConcurso	urlbase/equipossedeconcurso	3
cancelarEquipoSedeConcurso	urlbase/equipossedeconcurso/ {idequiposedeconc}	3

TOTAL DE PUNTOS: 110

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

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 31 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**.