La ejercitación está diseñada para que sea resuelta por módulos. Estos se encuentran ordenados para ser resueltos de manera incremental, y los mismos son dependientes, hay que resolverlos en orden.

Archivos a Entregar:

- **Código JAVA. En Repositorio de GitHub. No te olvides de agregar como colaboradora a la profe ☺ carolinadigitalhouse**
- **Un diagrama UML que contenga todas las relaciones que se presentan en el ejercicio.** No es necesario que declare los constructores, getters y setters en el diagrama UML. Usar la siguiente herramienta para entregar el UML:
    - **https://www.draw.io/**

Proyecto – JAVA
Para armar el proyecto utilizar spring boot: https://start.spring.io/ , además deberás implementar una **api rest.**

Fecha de entrega Límite:

Cualquier push al repositorio GIT luego de las 22hs. no será evaluado.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte A

1. Realizar un diagrama de clase que modele a la clase Alumno.
   En principio, un alumno posee un nombre **(String)**, un apellido **(String)** y un código de alumno **(Integer)**.

2. Implementar la clase creando los atributos necesarios.

3. Crear un constructor para el alumno que tome por parámetro un nombre, un apellido y un código de alumno.

4. Un alumno es igual a otro si sus códigos de alumno son iguales

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte B

1. Realizar un diagrama de clase que modele a la clase Curso. En principio, un curso posee un nombre **(String)** y un código de curso **(Integer)**.

2. Implementar la clase creando los atributos necesarios.

3. Crear los getter y setters para los atributos anteriores.

4. Un curso es igual a otro si sus códigos de curso son iguales

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte C

1. Realizar un diagrama de clase que modele a la clase Profesor. En principio, un profesor posee un nombre **(String)**, un apellido **(String)**, una antigüedad **(Integer)** y un código de profesor **(Integer)**.

2. Implementar la clase creando los atributos necesarios.

3. Crear los getter y setters para los atributos anteriores.

4. Un profesor es igual a otro si sus códigos de profesor son iguales

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte D

Se quiere agregar al modelo anterior dos nuevas categorías de profesores.

Los profesores titulares y los profesores adjuntos. Un profesor titular tiene una especialidad **(String)** y un profesor adjunto tiene una cantidad de horas que dedica para consultas **(Integer)**

1. ¿Cómo modificaría el diagrama de clase de Profesor que realizó anteriormente?

2. Modificar la implementación contemplando los nuevos cambios. Crear las clases que sean necesarias.

3. Crear los getters y setters necesarios para los nuevos atributos.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte E

Un curso además de tener un nombre y un código de curso, posee un profesor titular **(ProfesorTitular)**, un profesor adjunto **(ProfesorAdjunto)**, un cupo máximo de alumnos **(Integer)** y una lista de alumnos inscriptos **(List<Alumno>)**.

4. ¿Cómo modificaría el diagrama de clase de Curso que realizó anteriormente?
5. Modificar la implementación contemplando los nuevos cambios.
6. Crear los getters y setters necesarios para los nuevos atributos.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte F

1. Realizar un diagrama de clase que modele a la clase Inscripción.
   En principio, una inscripción tiene un alumno **(Alumno)**, un curso **(Curso)** y una fecha de inscripción **(Date)**.

2. Implementar la clase creando los atributos necesarios.

3. Crear un constructor de Inscripción que tome un alumno, un curso y construya una inscripción con la fecha del día. La clase Date permite utilizar fechas en Java. Para crear la fecha del día basta solo con especificar new Date().

   Ejemplo:

   Date fechaDelDia = new Date()

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte G

1. Realizar un diagrama de clase que modele a la clase DigitalHouseManager. En principio, DigitalHouseManager tiene una lista de alumnos **(List <Alumno>)**, una lista de profesores **(List <Profesor>)**, una lista de cursos **(List <Curso>)** y una lista de inscripciones **(List <Inscripcion>)**.

2. Implementar la clase creando los atributos necesarios.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte H

1. Crear un método en la clase **Curso** que permita agregar un alumno a la lista. El método devolverá **true** si el alumno puede agregarse o **false** en caso de que no haya cupo disponible.

    - **public Boolean agregarUnAlumno(Alumno unAlumno)**

2. Crear un método en la clase **Curso** que permita eliminar un alumno de la lista de alumnos del curso.

    - **public void eliminarAlumno(Alumno unAlumno)**

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte I

1. Crear un método en la clase **DigitalHouseManager** que permita dar de alta un curso. El método recibe como parámetros el nombre del curso, el código y el cupo máximo de alumnos que se admite. El método debe crear un curso con los datos correspondientes y agregarlo a la lista de cursos.

    - **public void altaCurso(String nombre, Integer codigoCurso, Integer cupoMaximoDealumnos)**

2. Crear un método en la clase **DigitalHouseManager** que permita dar de baja un curso. El método recibe como parámetro el código del curso. El método debe utilizar el código del curso para buscarlo en la lista de cursos y eliminarlo de la lista.

    - **public void bajaCurso(Integer codigoCurso)**

3. Crear un método en la clase **DigitalHouseManager** que permita dar de alta a un profesor adjunto. El método recibe como parámetros el nombre del profesor, el apellido, el código y la cantidad de horas disponibles para consulta. La antigüedad inicial del profesor será cero. El método debe crear un profesor adjunto con los datos correspondientes y agregarlo a la lista de profesores.

    - **public void altaProfesorAdjunto(String nombre, String apellido, Integer codigoProfesor, Integer cantidadDeHoras)**

4. Crear un método en la clase **DigitalHouseManager** que permita dar de alta a un profesor titular. El método recibe como parámetros el nombre del profesor, el apellido, el código y la especialidad. La antigüedad inicial del profesor será cero. El método debe crear un profesor titular con los datos correspondientes y agregarlo a la lista de profesores.

    - **public void altaProfesorTitular(String nombre, String apellido, Integer codigoProfesor, String especialidad)**

5. Crear un método en la clase **DigitalHouseManager** que permita dar de baja a un profesor. El método recibe como parámetro el código del profesor. El método debe utilizar el código del profesor para buscarlo en la lista de profesores y eliminarlo de la lista.

    - **public void bajaProfesor(Integer codigoProfesor)**

6. Crear un método en la clase **DigitalHouseManager** que permita dar de alta a un alumno. El método recibe como parámetros el nombre, el apellido y el código del alumno. El método debe crear un alumno con los datos correspondientes y agregarlo a la lista de alumnos.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

- **public void altaAlumno(String nombre, String apellido, Integer codigoAlumno)**

7. Crear un método en la clase **DigitalHouseManager** que permita inscribir un alumno a un curso. El método recibe como parámetros el código del alumno y código del curso al que se inscribe.

   - **public void inscribirAlumno(Integer codigoAlumno, Integer codigoCurso)**

   El método debe:

   - Buscar el curso al que se quiere inscribir.
   - Buscar al alumno al que se quiere inscribir.
   - Inscribir al alumno si es posible.
   - En caso de ser posible, crear una inscripción y setearla con los datos que corresponden.
     - Agregar la inscripción a la lista de inscripciones.
     - Informar por pantalla la inscripción realizada.
   - Si no hay cupo disponible:
     - Informar por pantalla que no se pudo inscribir porque no hay cupo

8. Crear un método en la clase **DigitalHouseManager** que permita asignar a un curso sus profesores. El método recibe como parámetros el código del curso, el código del profesor titular y el código del profesor adjunto

   - **public void asignarProfesores(Integer codigoCurso, Integer codigoProfesorTitular, Integer codigoProfesorAdjunto)**

   El método debe:

   - Buscar al profesor titular en la lista de profesores.
   - Buscar al profesor adjunto en la lista de profesores.
   - Asignarle al curso ambos profesores.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte J

1. Crear un Rest Controller para DigitalHouseManager en donde se pueda atender a la siguientes solicitudes:
2. http:localhost:8000/crearTitular que permita dar de alta un profesor titular.
3. http:localhost:8000/crearAdjunto que permita dar de alta un profesor adjunto.
4. http:localhost:8000/crearCurso que permita dar de alta un curso.
5. http:localhost:8000/asignarProfesor que permita asignarle un profesor titular y un adjunto a cada curso.
6. http:localhost:8000/altaAlumno que permita dar de alta a los alumnos.
7. http:localhost:8000/inscribirAlumno que permita Inscribir a dos alumnos en el curso de Full Stack por ejmplo
8. http:localhost:8000/bajaProfesor que permita dar de baja a un profesor
9. http:localhost:8000/detalleAlumno que permita obtener todos los datos de un alumno dado, un nombre de alumno en particular.
10. http:localhost:8000/actualizarAlumno que permita modificar los datos de un alumno en particular.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.

Parte K

1. ¿Cómo modificaría el diagrama de clases para que se le pueda consultar a un alumno a qué cursos está inscripto?

   // Responder a la pregunta en el rest controller, mediante un comentario.

ACLARACIÓN:

Como se está trabajando con listas, eliminando y agregando objetos, deberás implementar el equals a las clases que sean necesarias. Además, si querés podes implementar el toString() de estos objetos.