CIFP la laboral
CENTRO INTEGRADO DE FORMACIÓN PROFESIONAL

PROGRAMACIÓN – 1º DAM **PRUEBA 10 (6-ABRIL-2022) curso 2021/2022**	
Nombre y Apellidos:	**Puntuación:**

Se tiene en la url: https://github.com/luisdbb/federacion.git un proyecto Java implementado en Eclipse que responde al diagrama de clases y a las especificaciones del final de este documento y que contiene la solución a todos los ejercicios propuestos en las pruebas anteriores hasta el momento. Este proyecto servirá de base para la resolución de los ejercicios de esta prueba.

0. Se van a añadir 2 nuevas clases al diagrama de clases, `Patrocinador` y `Responsable`, junto a las relaciones entre sí y con el resto de entidades del diagrama. Deberán crearse estas clases de partida, así como todos los ejercicios que corresponda según el caso de la tabla siguiente:

		EVAL1				EVAL 2				EVAL 3			
Ejercicio	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
suspensos Eval1	X	X	X	X					X	X	X		
suspensos Eval2	X				X	X	X	X	X	X	X		
sólo Eval3	X				X				X	X	X	X	X
	-	0,5	1,5	1,5	0,5	1	1	1,5	1	2	2	2	2

1. (Máx 0,5ptos.) Implementar el método público estático `leerHora()` en la clase `Utilidades.java` para devolver un objeto de la clase `java.time.LocalTime` válido correspondiente a una hora del día concreta (HH:mm:ss, de 00:00:00 a 23:59:59).

2. (Máx 1,5ptos.) Completar las nuevas clases `Patrocinador` y `Responsable`, junto a las relaciones con el resto de entidades del diagrama de clases, de forma que:
   - Toda `Prueba` tiene obligatoriamente un `Patrocinador` y solo uno. Así mismo, cada `Patrocinador` tiene su propio y único `Responsable` (que es una `Persona`)
   - Las nuevas clases dispondrán de un constructor por defecto, otro de copia y otro constructor con todos los campos obligatorios, así como métodos getters y setters para cada campo siguiente:
     - un identificador propio y único para cada clase. Y, además,:
     - Para la clase `Patrocinador`:
       - Un nombre único que no se puede repetir, que es una cadena de caracteres de entre 3 y 150 caracteres, siendo válidos los alfabéticos (letras) o numéricos (dígitos) solamente.
       - Una URL con la cadena de caracteres de la web del patrocinador (si la hay).
       - El valor de la dotación en euros que realiza para patrocinar la/s prueba/s. Es obligatorio.
     - Para la clase `Responsable`:
       - El teléfono profesional, que es una cadena de caracteres de 9 o 10 dígitos. Es obligatorio.
       - La franja horaria del día en la que puede atender llamadas, con su hora de inicio y de fin.

- Implementar los métodos `nuevoResponsable()` y `nuevoPatrocinador()` para obtener sendos objetos completos valida dos desde la entrada estándar por parte de un usuario.

3. (Máx 1,5 ptos.) (apartado A) Modificar la clase `Prueba` para añadir su `Patrocinador` y, por tanto, actualizar los métodos adecuados de esta clase para adecuarlo al nuevo diagrama de clases:
    - Incluir los getters/setters para ese nuevo campo.
    - Añadir un nuevo constructor con este campo obligatorio.
    - Actualizar el método `nuevaPrueba()` para que solicite los datos de su patrocinador.
    - Actualizar el método `toString()` para que muestre el nombre del patrocinador.

    (apartado B:) Implementar el método `toString()` de la clase `Responsable.java` para que devuelva la cadena de caracteres con los datos del responsable de esta forma:
    <idResponsable> + '.' + <nombre> + '(' + <NIFNIE> + ')' + "horario de: " + <horaIni(HH:mm)> + " a " + <horaFin(HH:mm)> + " tfno: " + <telefonoProf>

    (apartado C:) Implementar en la clase `Patrocinador` los métodos: `mostrarBasico()` y `mostrarCompleto()` que devuelvan una cadena de caracteres con los datos del patrocinador, en el primer caso sólo el idPatrocinador + nombre + web (si la hay). En el caso completo incluir, además de los campos anteriores, la dotación en euros (`xx.xx euros`), así como los datos del responsable.

4. (máx 0,5ptos) Implementar el método `data()` para los responsables (clase `Responsable.java`) de forma que devuelva una cadena de caracteres con el siguiente orden:
    <idResponsable> | <idPersona> | <telefonoProf> | <horaIni(HH:mm)> | <horaFin(HH:mm)>

5. (Máx 1 pto.) Implementar la interfaz `Comparable` para la clase `Patrocinador.java`, de forma que se ordenen según su dotación en orden creciente y desempatar en función de la franja horaria de atención de su responsable (de mayor a menor, si la hay) y, por último, si sigue habiendo empate, deshacerlo por el valor creciente del campo `idPatrocinador`.

6. (Máx:1pto) Implementar una función estática en un nuevo fichero `PrincipalExam10.java` a la que se le pasa como argumento un array con una colección de objetos `Patrocinador`. La función recorrerá toda la colección y exportará hacia un fichero binario de nombre `patrocinadores.dat` los datos de todas los patrocinadores, ordenados según el ejercicio anterior, de uno en uno.

7. (Máx 1,5ptos.) Implementar una función estática en `Responsable.java` de nombre `importarResponsables()` que tome datos de responsables desde un fichero de caracteres de nombre `responsables.txt`. Este fichero contiene los datos de un `Responsable` de la forma establecida en su método `data()` (ejercicio 4), de forma que la función importará los datos de un responsable por cada línea en el fichero (utilizar el método estático de la clase `Datos.java::buscarPersonaPorId(long)` para obtener los datos del campo `Persona` del responsable). Con esos datos importados compondrá un conjunto de Responsables ordenados por el valor creciente de su idResponsable, que será posteriormente recorrido mediante un iterador para mostrar por la salida estándar los datos de cada responsable de la forma:
    <idResponsable> + ": " + <nombre> + '('<NIFNIE>')' + " tfno: "+ <telefonoProf> +'\n'

8. (Máx 1pto.) (apartado A) Implementar el método `data()` para los patrocinadores (clase `Patrocinador.java`) de forma que devuelva una cadena de caracteres con el siguiente orden:
    <idPatrocinador> | <idRepresentante> | <nombre> | <dotacion> | <web>

(apartado B) Crear en el programa principal de la clase `PrincipalExam10.java` una colección con los siguientes datos de patrocinadores:

idPatr	nombre	dotacion	web	idResp	tfnProf	horaINI	horaFIN	idPersona
1	ALSA	500.00	www.alsa.es	1	902422202	00:00	23:59	1011
2	Ayto. Gijón	250.00	www.gijon.es	2	985181105	09:00	18:00	1012
3	Universidad de Oviedo	350.50	www.uniovi.es	3	985103000	08:30	20:00	1013
4	CIFP La Laboral	255.99	www.cifplalaboral.es	4	985185503	08:30	18:00	1014

(utilizar el método estático de la clase `Datos.java::buscarPersonaPorId(long)` para obtener los datos del campo `Persona` del responsable del patrocinador)

9. (Máx 2ptos.) (apartado A) Crear las tablas adecuadas en la BD relacional `bdfederacion` para persistir los datos de las nuevas clases `Patrocinador` y `Responsable`, junto a sus restricciones correspondientes, tanto de tipo de dato como de valores válidos, unicidades, claves primarias y claves foráneas.
   (apartado B) Desde el programa principal de la clase `PrincipalExam10.java` implementar la funcionalidad para recorrer la colección del ejercicio 8 e insertar esos datos sobre las tablas correspondientes, a través de métodos `insertarConID(T)` y `insertarSinID (T)`, correspondientes a la implementación de la interfaz `operacionesCRUD<Patrocinador>`.

10. (máx 2ptos) Añadir a la interfaz `operacionesCRUD<T>` un nuevo método `T buscarPorID(long id)` que devuelve el objeto `T` cuyo identificador coincide con el valor que se pasa como parámetro (o `null` en caso de que no exista) haciendo una consulta a la tabla correspondiente a la entidad del tipo `T` en la BD `bdfederacion`. Reimplementar ese método para las entidades `Atleta` y `Patrocinador`, así como para la enumeración `Lugar`.

11. (Máx 2ptos.) (apartado A) Crear las tablas (junto a sus campos) correspondientes a las entidades `Atleta`, `Equipo`, `Lugar`, `Manager`, `Persona`,y `Prueba`, (además de las del ejercicio anterior) junto a sus restricciones de claves primarias y claves foráneas, etc.
    (apartado B) Implementar la interfaz `operacionesCRUD<Manager>` (incluyendo el método del ejercicio 10).

12. (Máx 2ptos.) Añadir a la clase `Prueba` el campo `Patrocinador` (si no estaba implementado ya) y preparar esta clase para que, tras llamar al método `Prueba.nuevaPrueba()` se inserten todos los datos mínimos obligatorios para una nueva prueba, estos son:
    - idPrueba (autoincremental)
    - nombre de la prueba
    - fecha de celebración y lugar
    - individual o por equipos
    - patrocinador (o bien uno nuevo o bien uno ya existente en la BD, a elegir por el usuario)

    Probar en la función principal del fichero `PrincipalExam10.java` la implementación anterior y ver que funciona correctamente, insertando los datos de una nueva prueba en las tablas que corresponda.

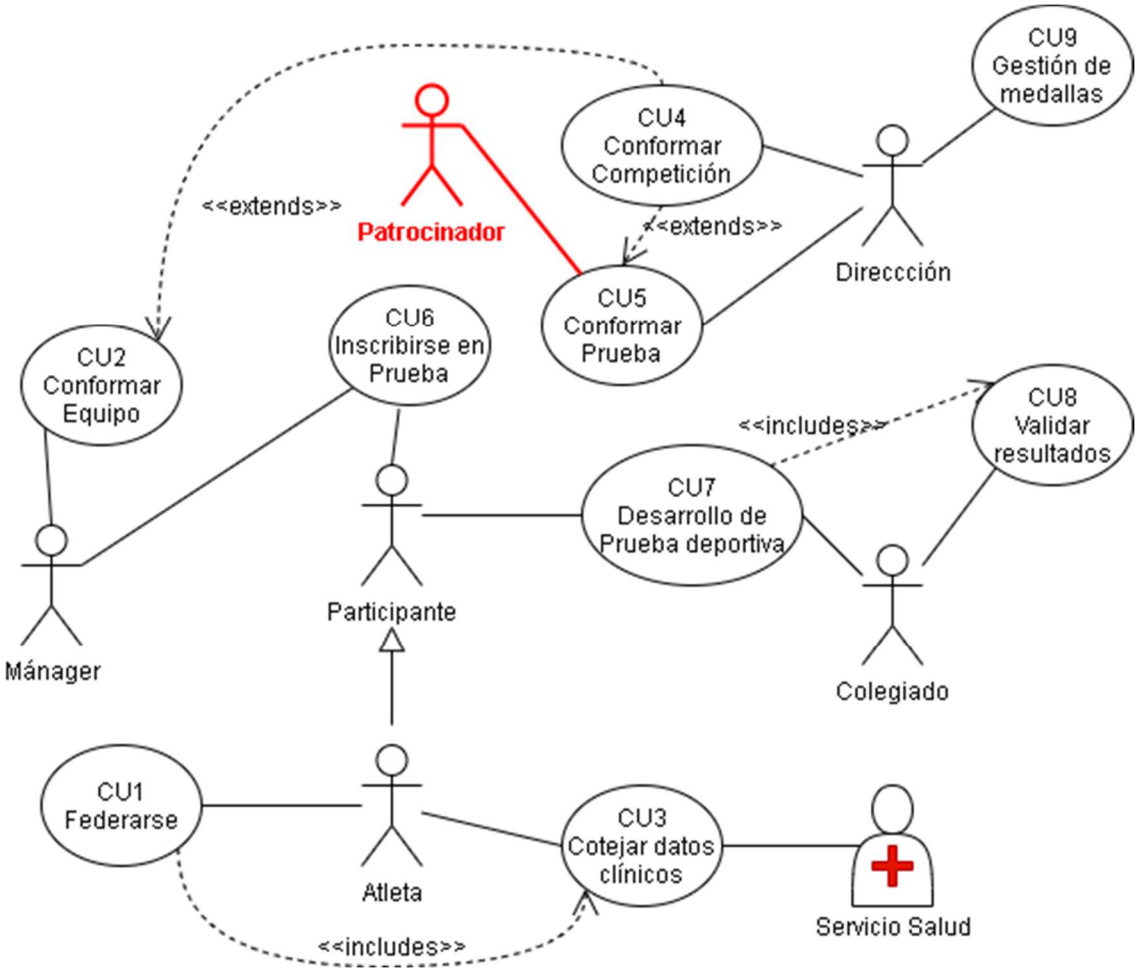

Ilustración 1Diagrama de casos de uso

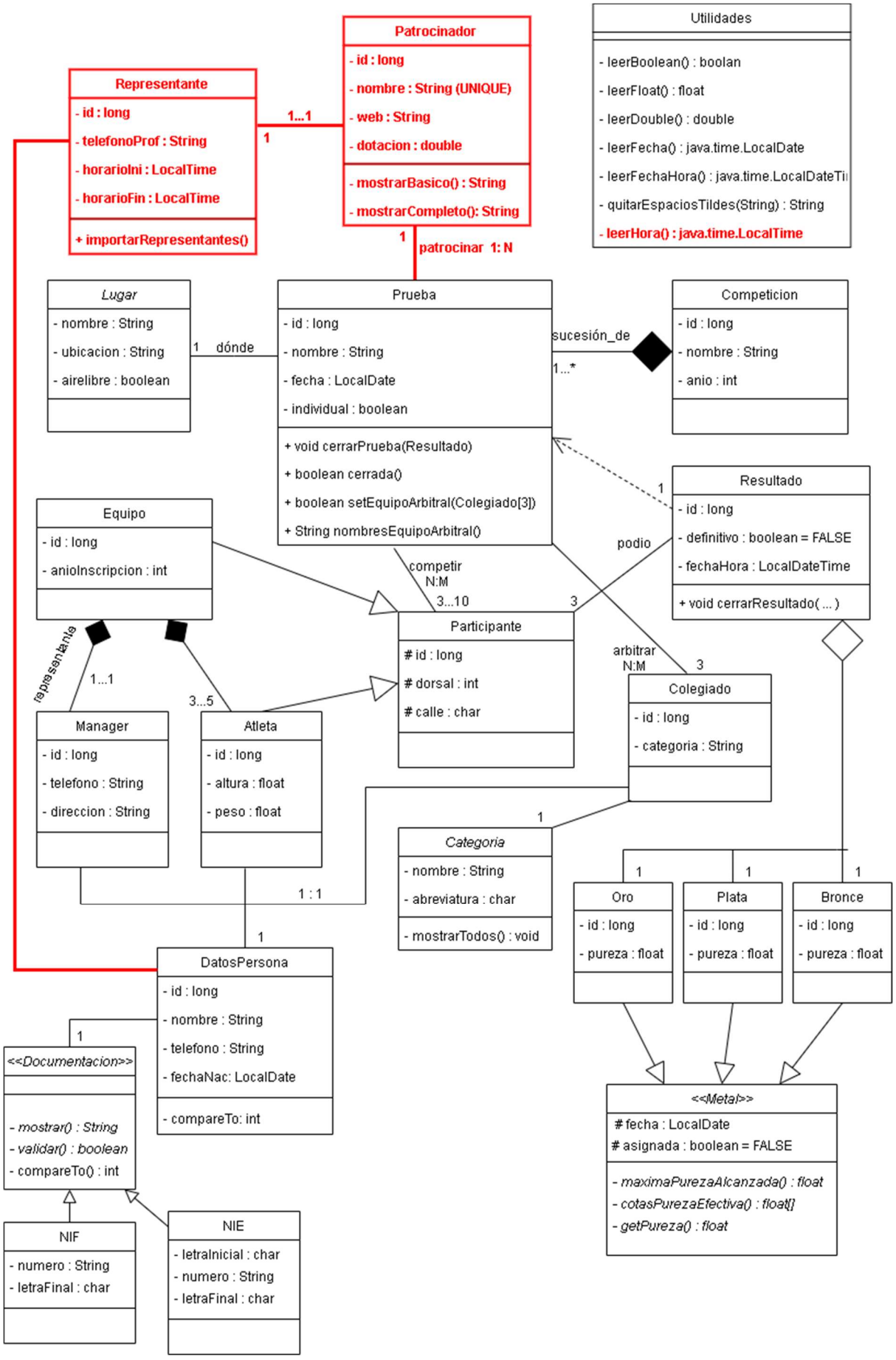

Ilustración 2Diagrama de clases

Descripción del sistema a modelar:

“Gestión de competiciones de una Federación deportiva”

El sistema informático permite a la dirección de la Federación manejar los datos de las competiciones a lo largo de su historia. Para conformar cada **competición,** se *identifica* de forma única en el sistema, y se guarda su *nombre* y el *año* en que se celebra. Toda competición es una sucesión de diferentes **pruebas** (al menos una), que van *identificadas* y se registran sus *nombres*, *fechas* y *lugares* de celebración de cada prueba. También se marca si se trata de una prueba *individual* o colectiva. Los **lugares** donde se llevan a cabo las pruebas siempre son los mismos: de ellos se tiene su *nombre*, *ubicación* y un campo que indica si es al *aire libre*.

En cada prueba siempre compiten entre 3 y 10 **participantes**, cada cual con su propio *identificador*, nº de *dorsal* (que siempre es un valor entre 001 y 150) y la letra de la *calle* por la que correrá. Si la prueba es individual, entonces cada participante se corresponderá con un **atleta.** Si no lo es, entonces la prueba se denomina colectiva y los participantes serán **equipos** conformados por varios atletas (mínimo 3 y máximo 5). Todos los equipos tienen su propio *identificador* de equipo y se guarda el *año* en que se inscriben, dado que un atleta podría formar parte de varios equipos distintos siempre y cuando sea en años de competición diferentes. Además, por cada equipo se tiene un único **mánager** (que se *identifica* unívocamente y se guarda tanto su *teléfono* como su *email*). Un mánager representa al mismo equipo siempre, pero no a los atletas individualmente. Los mánagers mandan la información de su equipo a la federación al principio de la temporada, cuando se conforman las competiciones. Para ello los propios atletas deben estar federados previamente. De los atletas interesa guardar su propio *identificador* de atleta y sus datos físicos: *altura*, expresada en metros, y *peso*, expresado en kilogramos (ambas unidades con 1 decimal). La información sobre los datos clínicos que cada propio atleta aporta al sistema al federarse se coteja con los del servicio de salud.

Por otro lado, la inscripción de un participante en una prueba individual la realiza el propio atleta, pero si es una prueba por equipos la realizará únicamente el mánager en representación de todos los miembros de ese equipo.

Por último, el desarrollo de las pruebas son controladas siempre por 3 **colegiados** distintos, cada uno de ellos tiene su *identificador* y un campo para marcar la *categoría*. Son los encargados de tomar las mediciones y de validar el **resultado** de cada prueba, marcándolo como *definitivo* en una *fecha y hora* determinadas. Todos los resultados van *identificados* y cada uno de ellos se compone de un **oro** (primer puesto en la prueba), una **plata** (2º puesto) y un **bronce** (3º puesto). De estos metales se lleva la cuenta de su *pureza* (valor decimal en %), además de su *identificador* propio. Se desea guardar cuál es la mayor pureza alcanzada de cada tipo de metal en todos los tiempos. Toda la gestión de las medallas recae también en la dirección de la federación deportiva.