CIFP la laboral
CENTRO INTEGRADO DE FORMACIÓN PROFESIONAL

PROGRAMACIÓN – 1º DAM **PRUEBA 11 (27-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 va a actualizar el diagrama de clases con el que se viene trabajando de la federación deportiva. Deberán crearse/modificarse algunas clases de partida, es decir, de forma obligatoria para todo el alumnado, así como todos los demás ejercicios que corresponda según el caso:

1. Implementar una nueva clase de nombre `Tiempo` que representa el tiempo obtenido por un `Participante` en una `Prueba`. Este dato se establece después de desarrollarse la prueba y lo hace el perfil de colegiado al cerrar los resultados de ésta.
   Contiene el número de horas, de minutos, de segundos y de centésimas de segundo que obtuvo el participante en la prueba, todos ellos son campos obligatorios. Implementar la nueva clase al completo: con sus constructores (por defecto y con todos los argumentos), getters/setters de cada campo, así como una reimplementación del método `toString()` para que se muestre la duración del `Tiempo` expresado como <horas>`:`<minutos>`:`<segundos>`,`<centesimas>

   (EVAL1) incluir un método público estático `nuevoTiempo()` para ingresar por la entrada estándar los valores para un objeto de la clase `Tiempo` y validarlo correspondientemente. En dicho método, si el usuario no introduce un valor para algún campo por la entrada estándar entonces tomarlo como si fuera 0. En la validación completa, el tiempo total introducido siempre ha de ser > 0h0m0,00s
   Crear después en un programa principal (`PrincipalExamen11.java`) un array de 10 objetos `Tiempo` tomados a partir de sucesivas llamadas a la función `nuevoTiempo()`. Luego, recorrer ese array y mostrar por la salida estándar sólo los tiempos introducidos que sean menores de 1h. Al mismo tiempo que se recorre el array, almacenar sobre otras variables el mejor y el peor de todos ellos. Mostrar los datos del mejor y del peor `Tiempo` tras recorrer el array.
   Sobre el array anterior, volver a recorrerlo desde el comienzo y, en aquellas posiciones en las que haya un `Tiempo` < 1h, preguntar al usuario si desea modificar el valor. En caso afirmativo, realizar una llamada al método `nuevoTiempo()` para obtener y actualizar esa posición del array.

   (EVAL2) implementar la interfaz `Comparable` para esta clase, y crear una nueva clase `ComparadorTiempos`, para que se ordenen de menor a mayor tiempo los objetos `Tiempo`. Implementar una función que tome como parámetro la ubicación de un fichero binario que contiene distintos objetos `Tiempo`. Recorrer ese fichero para obtener la colección de los tiempos almacenados en el mismo y almacenarlos en una lista. Posteriormente, ordenar la lista de acuerdo

a los comparadores anteriores y recorrerla mediante un iterador. Aquellos tiempos que sean mayores de 1h eliminarlos de la lista. Para el resto, redondear el `Tiempo` de forma que un valor >50 en las centésimas incremente el valor de los segundos en 1 unidad (y luego siempre establecer a 0 las centésimas).

(EVAL3) diseñar una GUI para la inserción de un nuevo `Tiempo`, a través de un panel reutilizable que incluya los componentes necesarios para tomar todos los datos.

2. Actualizar la clase `Participante` para que incorpore los siguientes nuevos campos:
   - un campo `Tiempo`.
   - un campo booleano `penalizacion` que indica si hubo penalización o no por parte del participante. Por defecto se establece al valor `false`. Cuando este campo cambia a valor `true` entonces el próximo campo (otros) se hace obligatorio.
   - Un campo `otros` que es un campo descriptivo de no más de 500 caracteres que detalla el porqué de una penalización (si la hay)
   - Será necesario modificar la clase `Participante` en relación a sus constructores, métodos getters y setters para los nuevos campos, método `toString`()…

   (EVAL1) Implementar además el método `nuevoParticipante()` para obtener por parte de un usuario un objeto completo `Participante` desde la entrada estándar y validarlo adecuadamente de acuerdo a las especificaciones. Recuérdese que, 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, además de los nuevos campos introducidos.

   (EVAL2) (apartado A) Implementar la interfaz `Comparable` para la clase `Prueba.java`, de forma que se ordenen según su fecha, de más reciente a más antigua, y desempatar en función de si es individual (en cuyo caso se considera anterior a una de tipo colectiva) o por equipos. Por último, si sigue habiendo empate, deshacerlo por el valor del campo nombre en orden alfabético creciente.
   Implementar una función que tome como parámetro una Lista de pruebas, ordenarla según el criterio anterior y recorrer todos sus elementos. Con ellos, se construirá un diccionario<Lugar, Set< idPruebas>> en el que se incluirán sólo los identificadores de aquellas pruebas que sean individuales, estableciendo una entrada para cada Lugar con los identificadores de las pruebas que se celebran en dicho lugar.

   (EVAL2) (apartado B) Implementar la interfaz `Comparable` para la clase `Participante.java`, de forma que se ordenen según su tiempo en orden creciente y desempatar en función de la calle en la que participó (de menor a mayor) y, por último, si sigue habiendo empate, deshacerlo por el valor creciente del campo `idParticipante` .

   (EVAL3) (apartado A) Crear las tablas (junto a sus campos) en la BD necesarias para persistir toda la información de las entidades `Metal`, `Oro`, `Plata`, `Bronce`, `Participante` y `Resultado`, junto a sus restricciones de claves primarias y claves foráneas, etc.
   (apartado B) Crear las clases `DAO` necesarias para poder manejar con cualquier tipo de Metal las funciones de la interfaz `operacionesCRUD`.

   (EVAL3) (apartado A) Crear una `GUI` adecuada a modo de formulario para que el usuario inserte todos los datos mínimos obligatorios para una nueva `Prueba`, estos son:
   - idPrueba (automático, no editable)

- nombre de la prueba (entre 5 y 150 caracteres)
- fecha de celebración (al menos un mes posterior a la fecha actual)
- lugar (de entre los que ya existen en la BD)
- individual o por equipos
- patrocinador (uno ya existente en la BD, a elegir por el usuario)

Incluir 2 botones, "*Aceptar*" y "*Cancelar*" en la parte inferior del formulario.

(apartado B) Al pulsar el botón "*Aceptar*" se tendrá un `Listener` adecuado como función manejadora de dicho `Evento`. Desde ahí implementar el mecanismo adecuado para que tome cada uno de los valores desde los componentes utilizados en el formulario, los valide adecuadamente y construya con ellos un nuevo objeto `Prueba`. Este objeto, si es válido, se insertará en la BD en la tabla `pruebas` (solo con esos campos de momento) a través de la clase `PruebaDAO` que implementa la interfaz `operacionesCRUD<Prueba>. ()`

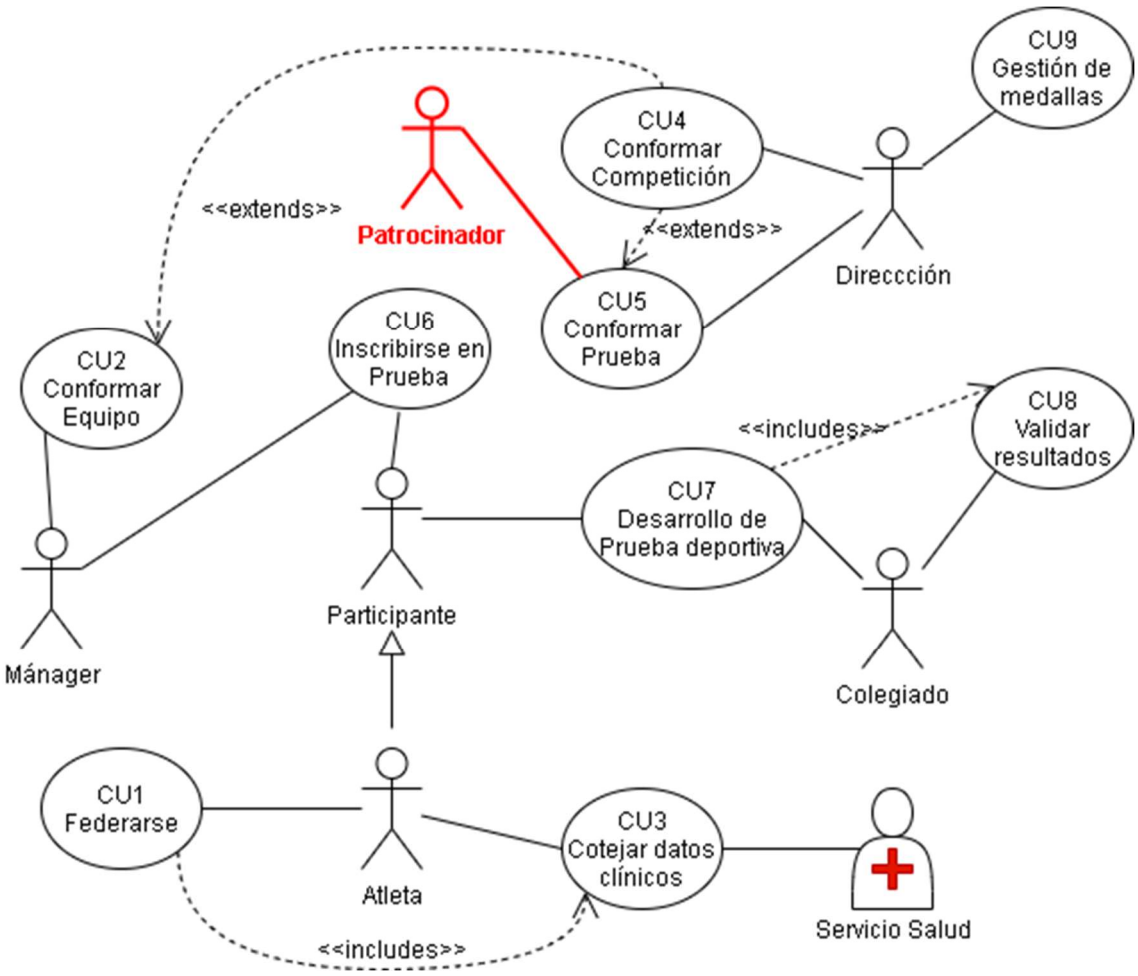

Ilustración 1Diagrama de casos de uso

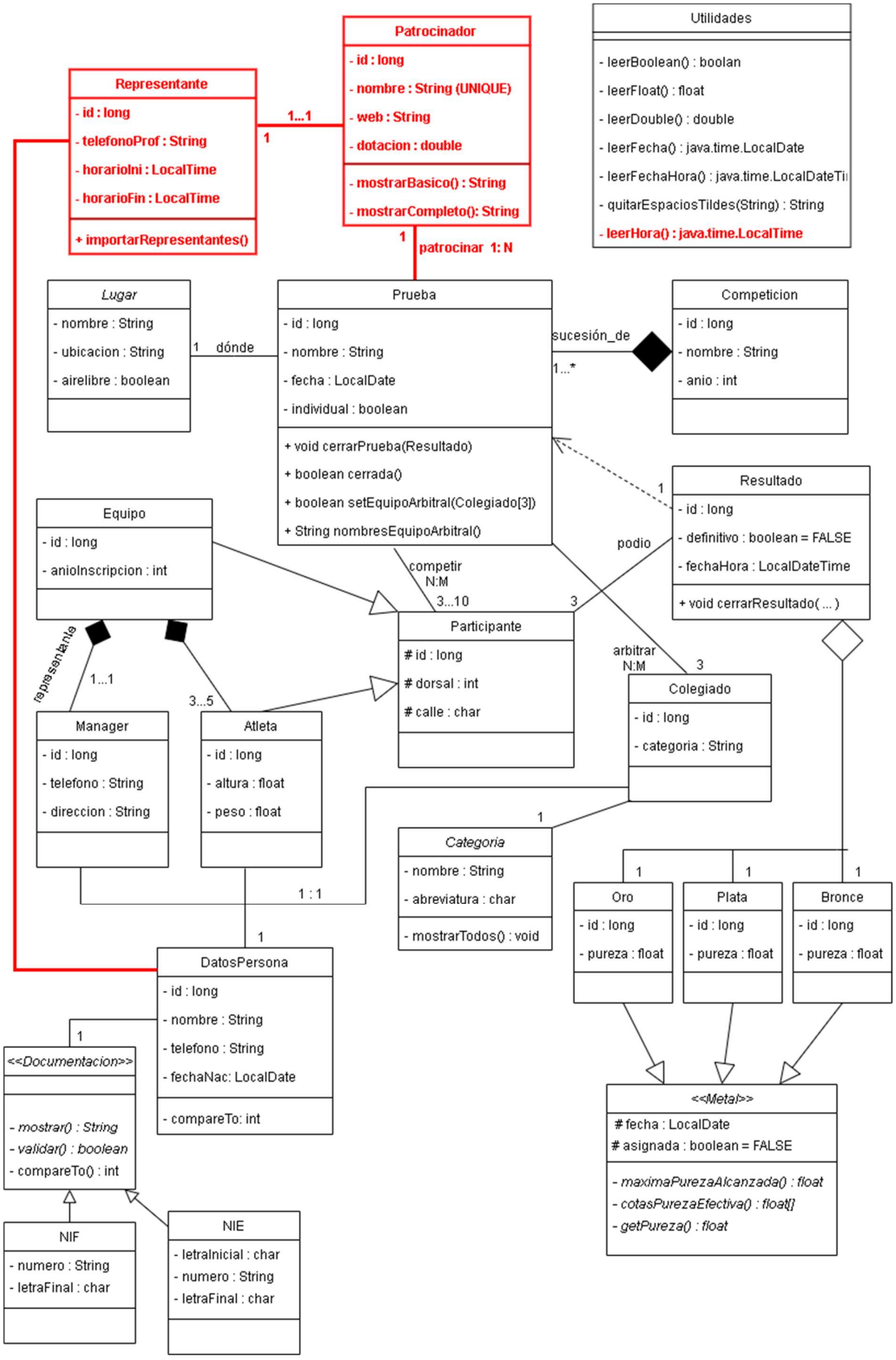

Ilustración 2Diagrama de clases

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.