Apartado A: Clase Persona

Para cada persona necesitaremos conocer su nombre, su dirección y su DNI. Todos estos atributos serán privados y de tipo `String`. También tendrá el atributo sexo (boolean) el cual servirá para indicar si esa persona es chico o chica. Este atributo será protegido.

Aparte del constructor habitual, esta clase debe tener también un constructor de copia y todos los métodos Get y Set necesarios para poder usar sus propiedades privadas.

Implementa también el método `toString` para que muestre los datos de una persona siguiendo el formato:

```
Nombre - DNI
Es un chico o Es una chica
Dirección
```

Apartado B: Clase Jugador

Esta clase va a representar a un jugador de un equipo de baloncesto. Como es obvio, todo Jugador es una Persona.

Un jugador tendrá los atributos privados altura (en metros) y posición. Los valores posibles que puede contener esta última propiedad son: *Base, Escolta, Alero, Ala-pivot* y *Pivot.*

También es necesario saber el numero de partidos que ha jugado el jugador (número entero positivo) y si está jugando en este momento o no. Ambos atributos privados.

Como es obvio, todo jugador empieza con 0 partidos jugados.

Para saber si está jugando o no, crea el atributo que creas conveniente. Al crearse, un jugador no debe estar jugando.

El constructor de la clase debe pedir todos los valores necesarios y asegurarse de que se introduce una altura mayor a 1.65 metros y un valor correcto en la propiedad posición.

Si la altura no es correcta, se colocará el valor 1.70 metros por defecto. Si el valor de la posición no es válido, se tomará *"Alero"* como posición por defecto para ese jugador.

Un jugador también tendrá un método llamado `jugando`, el cuál no tendrá argumentos y mostrará por pantalla si el jugador está en jugando en ese momento o no usando alguna de las siguientes frases:

```
El jugador nombre del jugador no está jugando.
El jugador nombre del jugador está jugando.
La jugadora nombre de la jugadora no está jugando.
La jugadora nombre de la jugadora está jugando.
```

Dependiendo de si es chico o chica y de si está o no jugando.

El método `toString` de esta clase debe mostrar los datos del jugador según el siguiente formato:

```
Nombre - DNI
Es un chico o Es una chica
Dirección
Altura: altura
Su posición es: posicion
Jugador/Jugadora en partidos_jugados partidos.
Ahora mismo está jugando o Ahora mismo no está jugando.
```

Apartado C: Clase Árbitro

Esta clase representa a un árbitro capacitado para dirigir un partido de baloncesto Como es obvio, todo Árbitro es una Persona.

De un árbitro necesitamos conocer su sueldo por partido (numero decimal entre 800.0 y 1200.0) y especialidad. Ambos atributos serán privados.

Los posibles valores para la especialidad de un árbitro son: *Principal, Auxiliar y Oficial de Mesa*.

El constructor de esta clase debe pedir todos los valores necesarios para crear un árbitro y asegurarse de que tanto al sueldo como a la especialidad se le asignan valores válidos. De no ser así, se colocará el valor 914.0 como sueldo por defecto y/o el valor *'Auxiliar'* como especialidad por defecto.

El método `toString` de esta clase debe mostrar los datos del árbitro según el siguiente formato:

```
Nombre - DNI
Es un chico o Es una chica
Dirección
Especializado/Especializada como especialidad que cobra sueldo€
por partido.
```

Apartado D: Clase Equipo

Esta clase representa un equipo de baloncesto.

De un equipo de baloncesto debemos conocer su nombre, su año de fundación, el número de victorias que lleva y su presupuesto en euros. Además, todo equipo tendrá, como es normal, una lista de jugadores y/o jugadoras[2]. Todos deben ser atributos privados.

Cuando se crea un equipo debe indicarse su nombre, su año de fundación, su presupuesto en euros y el número máximo de jugadores que va a tener. Es decir, al crearse el equipo se sabe el número de jugadores pero no qué jugadores son. Usa la estructura que creas conveniente para gestionar esto e inicializalá correctamente.

Para que el equipo sea válido debe tener mínimo 6 y máximo 12 jugadores/jugadoras. También debe tener un presupuesto mayor a 19.700€

El constructor debe comprobar que se cumplen las condiciones anteriores. De no ser así, usarán los siguientes valores por defecto: 8 jugadores/as y/o 20.000€ de presupuesto.

Como es obvio, al crearse un equipo tendrá 0 victorias.

Por otro lado, la clase tendrá los siguientes métodos:

- `aniadirJugador(Jugador nuevo)`: añade el jugador que se pasa como parámetro al equipo si es posible. Devuelve una cadena indicando si ha sido posible añadirlo o no.
- `enEquipo(Jugador player)`: devuelve Verdadero o Falso dependiendo de si el jugador que se le pasa como parámetro está en el equipo o no.
- `eliminarJugador(Jugador borrar)`: elimina del equipo el jugador que se pasa como parámetro si este está en el equipo. Devuelve una cadena indicando si ha sido posible realizar la acción o no.

---

2 Los equipos que vamos a programar pueden ser mixtos, es decir, no hay problema en que haya chicos y chicas en el mismo equipo.

- `inscripcion()`: devuelve Verdadero si hay hueco para apuntar nuevos jugadores al equipo o Falso en caso contrario.
- Todos los GET y SET que falten por añadir de las diferentes propiedades.

Finalmente, crea el método `toString` usando el siguiente esquema:

```
Nombre del equipo
Fundado en año_fundación. Con presupuesto€ de presupuesto actual.
Victorias: victorias_totales.

Integrantes del equipo:
==============================
datos del primer jugador/a
----------------------------------
datos del segundo jugador/a
-----------------------------------
. . .
-----------------------------------
Datos del último jugador/a.
```

Apartado E: Partido

Esta clase representa un partido de baloncesto.

De cada partido necesitamos conocer la fecha en la que se disputa[3], el nombre del pabellón en el que se juega y dos equipos: uno local y otro visitante. También es necesario que el partido tenga un árbitro asignado (*sí, en estos partidos solo es necesario que haya un árbitro. No le des más vueltas*). No puedes añadir más propiedades a esta clase. Si necesitas hacerlo, habla con el profesor.

La clase tendrá dos constructores:

- constructor que solo necesita la fecha y el nombre del pabellón. Tanto los equipos como el arbitro quedarán vacíos. Este partido no podrá jugarse hasta que no haya 2 equipos y un árbitro (ver más adelante).
- Constructor con todos los argumentos necesarios. Este partido podrá jugarse sin problemas (ver más adelante).

3 Puedes usar tipo String para facilitar las cosas o usar el objeto Date de Java para mejorar el ejercicio.

Los métodos de esta clase serán los siguientes:

- `aniadirLocal(Equipo local)`: añade o reemplaza el equipo local por el que se le pasa como parámetro.
- `aniadirVisitante(Equipo visitante)`: igual que el método anterior pero para el equipo visitante del partido.
- `aniadirArbitro(Arbitro nuevo)`: añade o reemplaza al arbitro del partido por el que se le pasa como parámetro.
- `jugarPartido()`: este método debe devolver una cadena. Si funcionamiento es el siguiente:
  - Un partido puede jugarse sí y sólo sí existen los dos equipos y hay árbitro asignado. En caso de que esto no se cumpla se devolverá la cadena *"El partido no puede jugarse porque no cumple con las condiciones mínimas."*
  - Si el partido puede jugarse, se realizará lo siguiente:
    - se incrementará en 1 el valor de la propiedad "partidos jugados" de cada jugador/a que disputa el partido.
    - Se decidirá al azar que equipo gana y se incrementará en 1 la propiedad 'victorias' de ese equipo.
    - A continuación se devolverá la cadena: *"El partido se ha disputado y ha ganado el equipo nombre_del_equipo ganador"*
- Todos los GET y SET que falten por añadir de las diferentes propiedades.

Apartado F: Diagrama

Haciendo uso de la herramienta que quieras, crea el diagrama de clases UML del diseño que has elegido para resolver esta práctica.

Adjunta el diagrama como un archivo de imagen dentro del paquete donde tienes los archivos de código.