



**POLITÉCNICO  
ESTELLA**

**ASIGNATURA: ETORNOS DE DESARROLLO**

**GRADO SUPERIOR EN DESARROLLO DE  
APLICACIONES MULTIPLATAFORMA**

**TAREA 10 TEMA 05**

Presentado por: Eugen Moga

## ÍNDICE

Ejercicios 1 .....	1
Ejercicios 2 .....	4
Ejercicios 3 .....	10

## Ejercicios 1

Se pretende realizar un programa basado en programación orientada a objeto que modele la conformación de un equipo de fútbol, según las siguientes especificaciones mínimas (se pueden añadir más si se desea):

Un **equipo** de fútbol, definido por su **nombre y país**, lo constituyen **técnicos** y **jugadores**. Estos últimos se diferencian según su rol (**portero, defensas, mediocampos y delanteros**). Todas las **personas** del equipo (técnicos y jugadores) tienen un **nombre, apellidos y edad**. Se informan para los técnicos sus **años de experiencia** y el hecho de ser **nacionales o extranjeros**. Se informan para los porteros la cantidad de **goles recibidos**. Para los demás jugadores se informa el número de **asistencias realizadas y la cantidad de goles anotados**. Todos los jugadores cuentan con un número de dorsal y la información para saber si **es titular o no**.

Define un nombre del paquete, las **clases** (en particular su tipo), sus **atributos** y métodos necesarios (justificando visibilidad y definiendo parámetros), las asociaciones entre clases, sus cardinalidades,...

Se requieren métodos constructores para los jugadores, técnicos y equipo, así como los métodos de encapsulación correspondientes a los atributos de cada clase. Se pueden añadir los métodos que se intuyen convenientes para este programa.

**Nombre del paquete:** modelo.futbol

**Clase Equipo:**

**Atributos:** nombre, país, List<Técnico> técnicos, List<Jugador> jugadores (Visibilidad privada -)

**Métodos:**

Constructor: public Equipo(String nombre, String pais)

Encapsulación: metodos getter y setter para el nombre y el país.

Método: agregarTécnicos y agregarJugador

Método: mostrarInformación del equipo

Visibilidad de los métodos y el constructor públicos.

**Clase Persona:** clase abstracta

**Atributos:** nombre, apellidos y edad (Visibilidad privada -)

**Métodos:**

Constructor: public Persona (String nombre, String apellidos, int edad)

Encapsulación métodos getter y setter para los tres atributos

Visibilidad de los métodos y el constructor públicos.

**Clase Técnico** hereda de la clase Persona

**Atributos:** añosExperiencia y esNacional (Visibilidad privada -)

**Métodos:**

Constructor: public Tecnico (String nombre, String apellidos, int edad, int añosExperiencia, boolean esNacional)

Encapsulación: métodos getter y setter para los dos atributos de la clase

Método mostrarInformacion para mostrar información del técnico.

Visibilidad de los métodos y el constructor públicos.

**Clase Jugador** clase abstracta y hereda de la clase Persona

**Atributos:** numeroDorsal, esTitular (Visibilidad privada -)

**Metodos:**

Constructor: public Jugador (String nombre, String apellidos, int edad, int numeroDorsal, boolean esTitular)

Encapsulación: métodos getter y setter para los atributos numeroDorsal y esTitular

Método: mostrarInformacionJugador para mostrar la información del jugador.

Visibilidad de los métodos y el constructor públicos.

**Clase Portero** hereda de la clase Jugador

**Atributos:** golesRecibidos (Visibilidad privada -)

**Metodos:**

Constructor: public Portero (String nombre, String apellidos, int edad, int numeroDorsal, boolean esTitular, int golesRecibidos)

Encapsulación: métodos getter y setter para el atributo golesRecibidos.

Método: mostrarInformacionPortero para mostrar información del portero

Visibilidad de los métodos y el constructor públicos.

**Clase JugadorCampo** es una clase abstracta y hereda de la clase Jugador

**Atributos:** asistencias, golesAnotados (Visibilidad privada -)

**Métodos:**

Constructor: public JugadorCampo (String nombre, String apellidos, int edad, int numeroDorsal, boolean esTitular, int asistencias, int golesAnotados )

Encapsulación: métodos getter y setter para los atributos asistencias y golesAnotados.

Método: mostrarInformacion para ver información

Visibilidad de los métodos y el constructor públicos.

**Clases Defensa, Centrocampista y Delantero:** heredan de la clase JugadorCampo, estas clases las utilizo para diferenciar los roles de los jugadores y no requieren atributos propios.

Asociaciones entre clases y Cardinalidades

**Equipo:**

Relación: Equipo tiene una relación de composición con Técnico y Jugador.

Cardinalidad:           1:N (1 equipo tiene muchos técnicos)  
                              1:N (1 equipo tiene muchos jugadores)

**Persona:**

Relación: Es una clase abstracta que generaliza (herencia) a Técnico y Jugador

Cardinalidad: 1 Persona puede ser 1 Técnico o 1 Jugador no hay multiplicidad en estas relaciones ya que son relaciones de herencia.

**Jugador:**

Relación: es una clase abstracta que generaliza a Portero y JugadorCampo

Cardinalidad: no hay multiplicidad ya que es una relación de herencia.

**JugadorCampo:**

Relación: es una clase abstracta que generaliza a Defensa, Centrocampista y Delantero

Cardinalidad: no hay multiplicidad ya que es una relación de herencia

Justificación de visibilidad.

Los atributos propios de cada clase los establezco como privados para garantizar el encapsulamiento y controlar el acceso a los datos

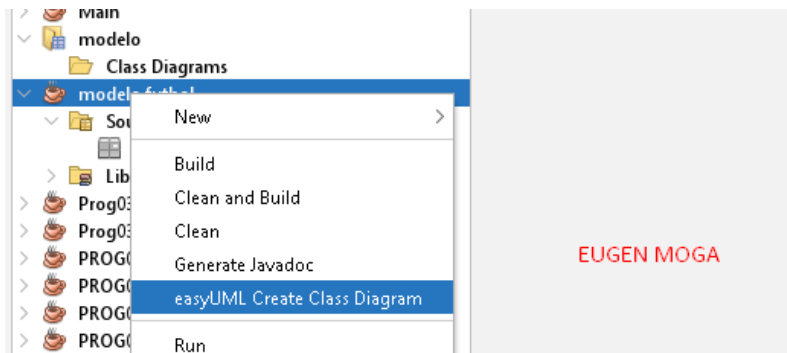
Visibilidad de los métodos y el constructor de cada clase públicos.

## Ejercicios 2

Deduce el diagrama de clases correspondiente y génalo con easyUML.

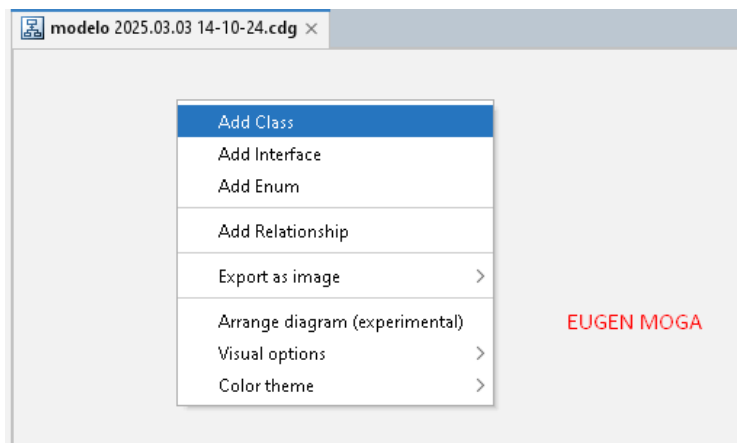
Primero creo el proyecto java “modelo.futbol” en netbeans sin ninguna clase y seguido creo un proyecto UML y le pongo el mismo nombre model.futbol

Una vez creados los proyectos java y UML le doy clic derecho en el proyecto modelo.futbol y le doy a easyUML Create Class Diagram

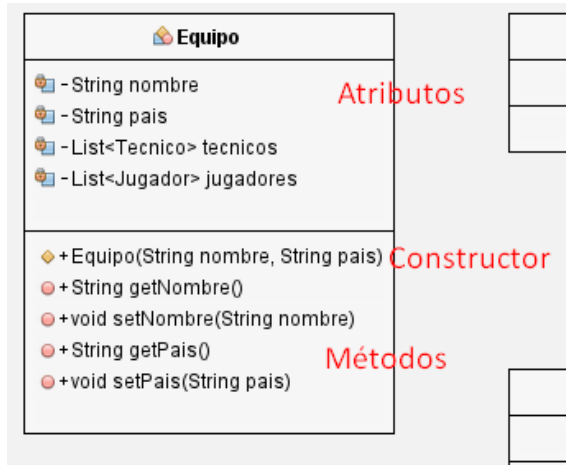


Automáticamente se genera un archivo .cdg en blanco que es donde voy a ir creando las clases con sus atributos y sus métodos, que previamente he definido en el ejercicio 1

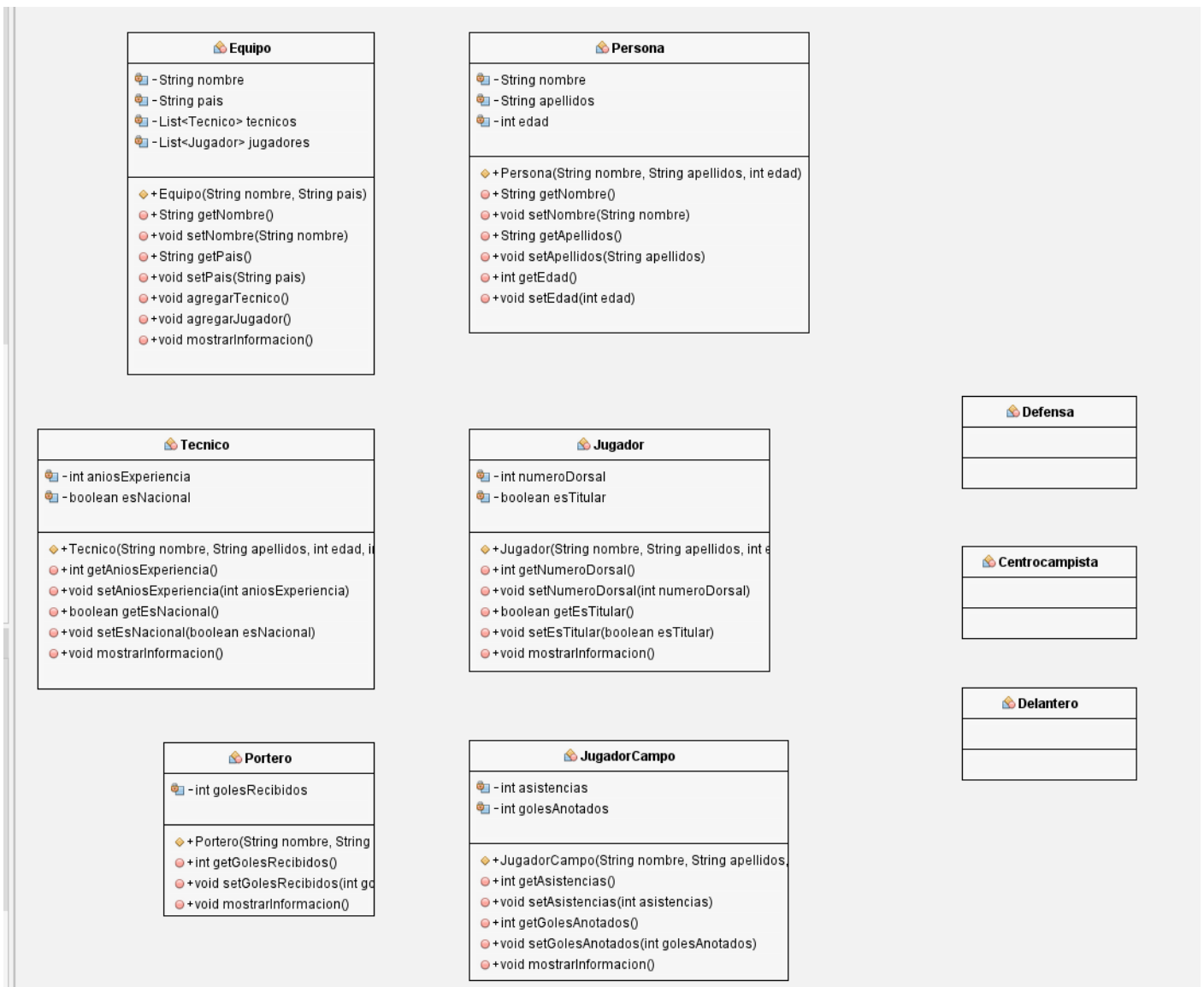
En el archivo .cdg le doy clic derecho y Add Class para agregar las clases



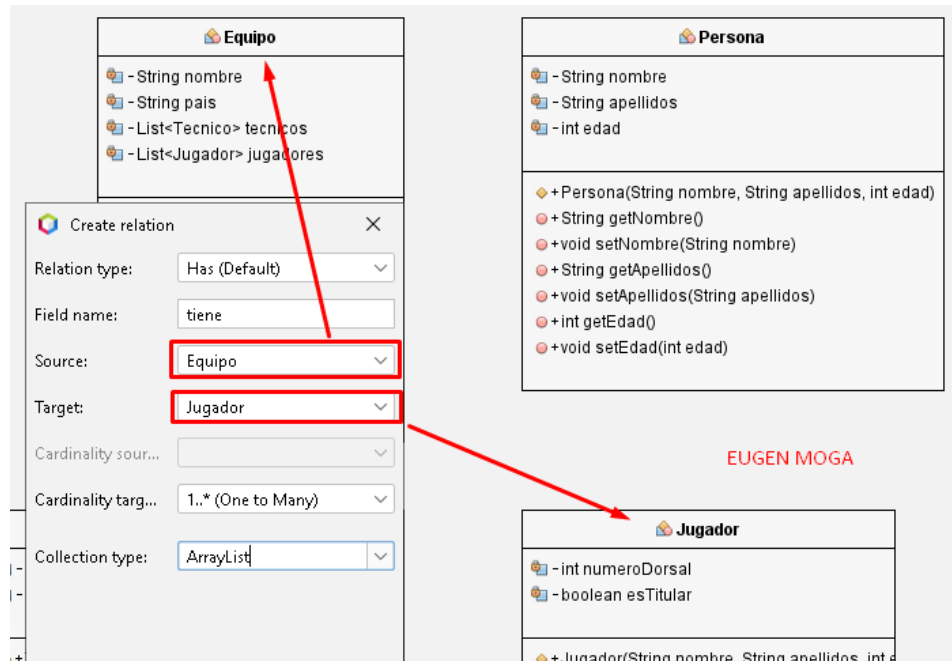
Una vez creadas todas las clases paso a configurar los atributos y los métodos establecidos para cada clase.



Al terminar todas las clases con sus atributos quedan de la siguiente manera

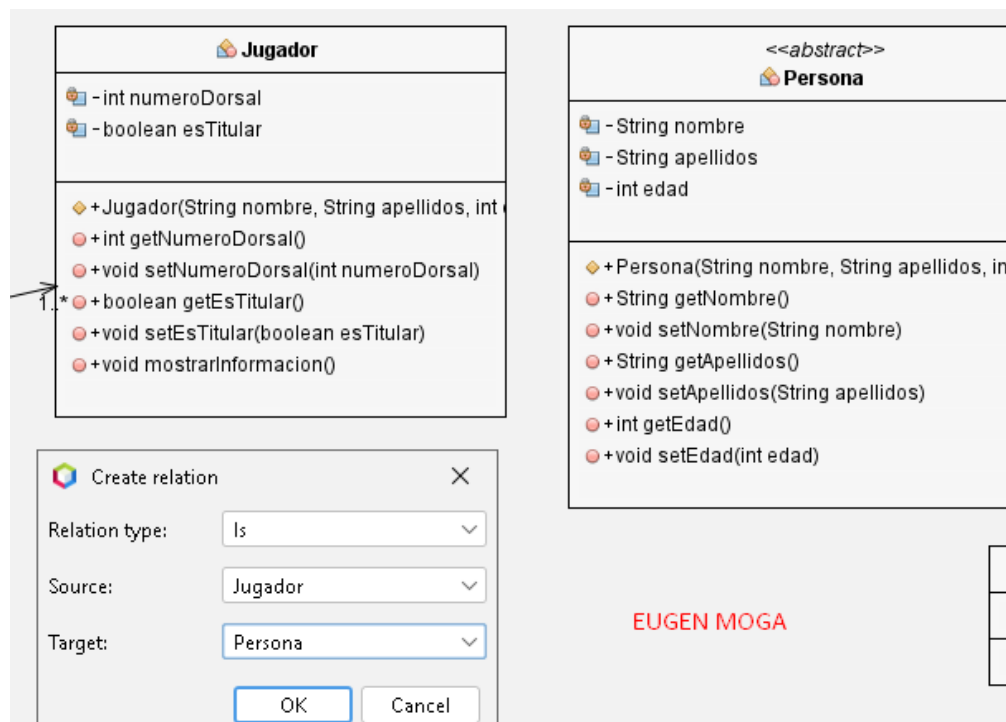


Ahora voy a poner las relaciones y las cardinalidades entre las clases, para ello le doy click derecho en un espacio vacío del diagrama Add Relationship en Relation type le pongo “Has composition” en Field name le pongo “tiene” porque un equipo tiene varios jugadores, en Source selecciono la tabla Equipo y en Target selecciono la tabla Jugador y en la Cardinality selecciono One to Many (uno a muchos)



Hago lo mismo entre Equipo y Técnico

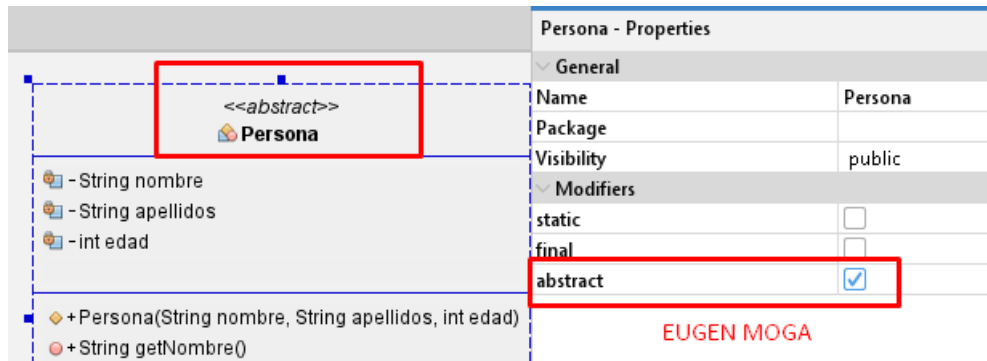
Para indicar que la clase Jugador hereda de la clase persona agrego la relación Is, de esta manera.



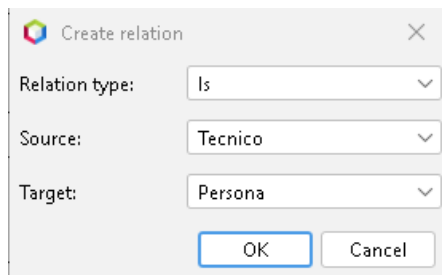


Hago lo mismo para Técnico le indico que hereda de la clase Persona a través de la relación Is.

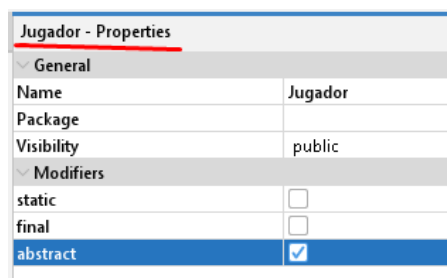
Para indicar que la clase Persona es abstracta selecciono la clase y en la barra de la derecha Propiedades selecciono la casilla abstract



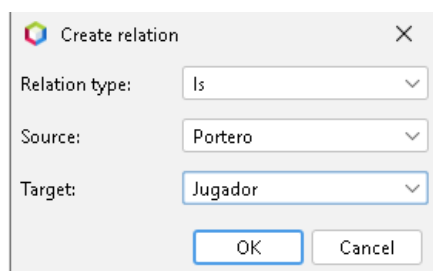
También creo la relación de herencia entre la clase Técnico y la clase Persona



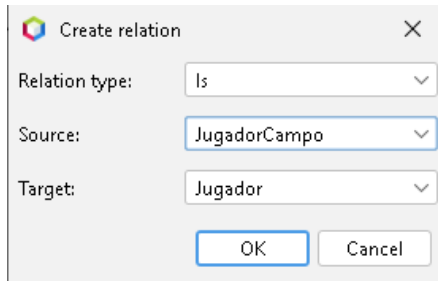
La clase Jugador también la marco como abstracta ya que no me interesa instanciar o objeto directamente de la clase jugador



Para finalizar hay dos clases de jugadores diferentes que se diferencian por la clase Portero que hereda de la clase Jugador y los jugadores de campo que también heredan de la clase Jugador



La clase JugadorCampo también la marco como abstracta y hereda de la clase Jugador



Create relation

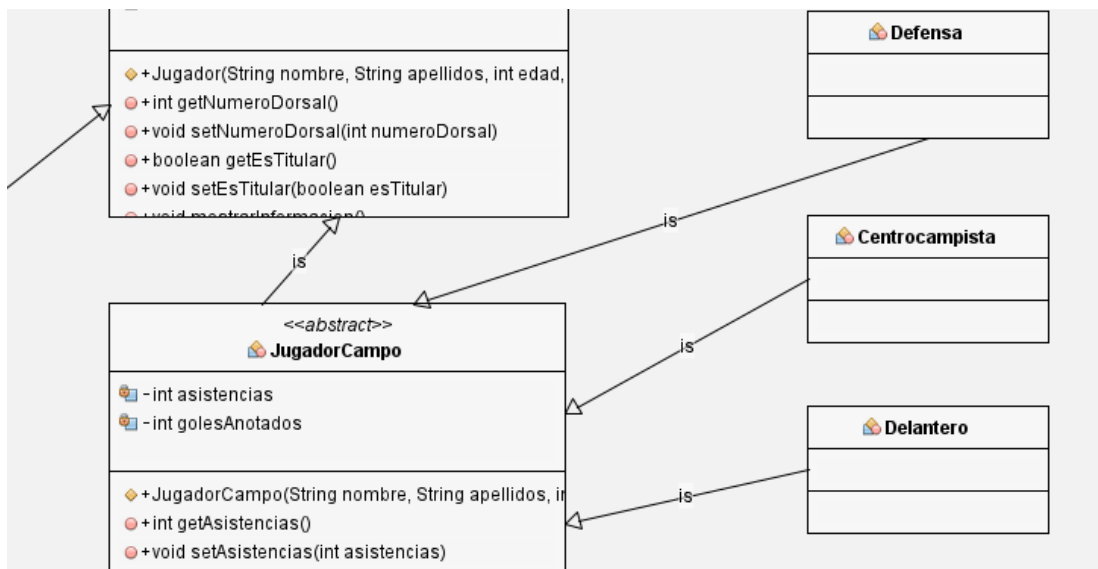
Relation type: Is

Source: JugadorCampo

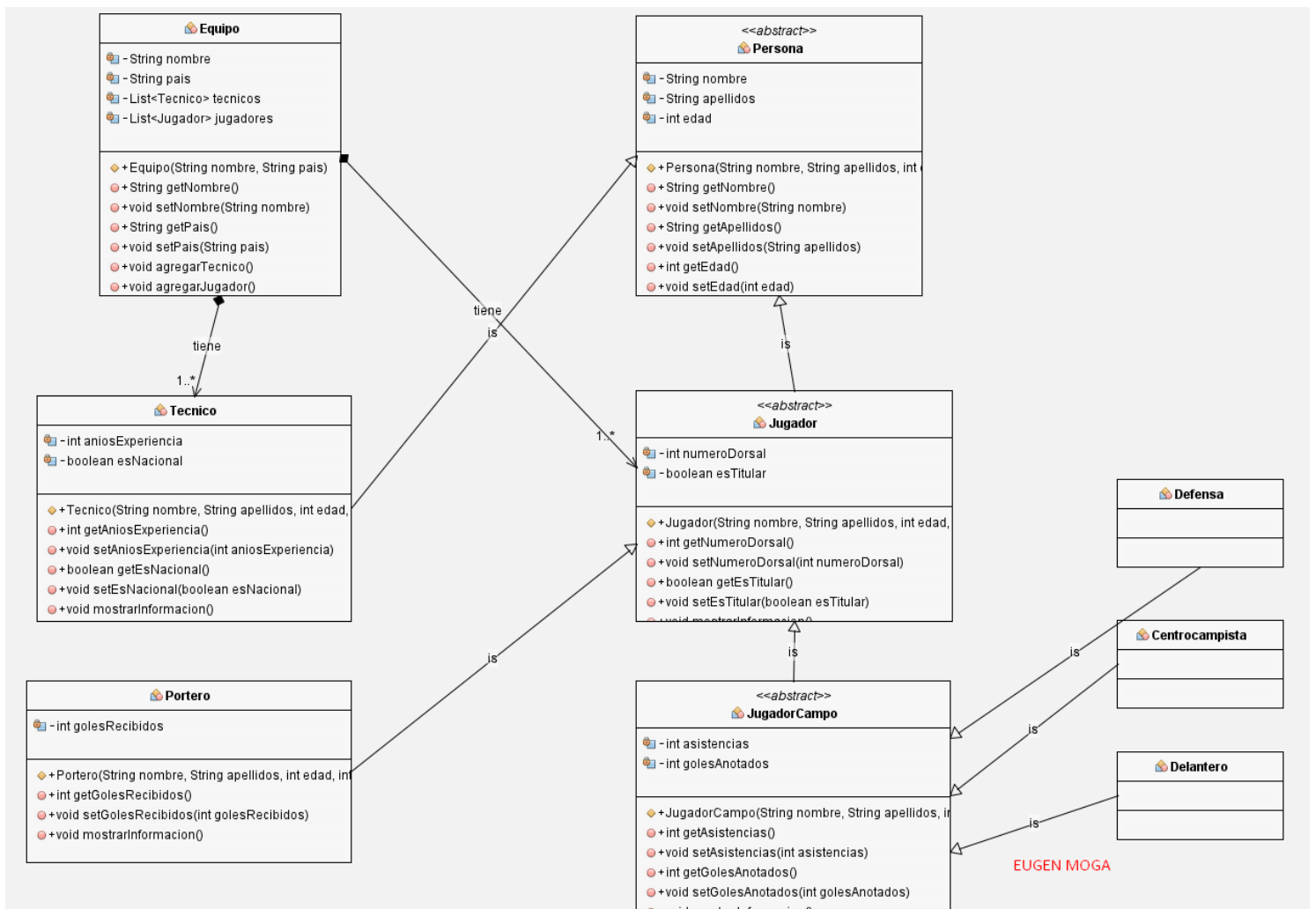
Target: Jugador

OK Cancel

Por ultimo la clase Defensa, Centrocampista y Delantero le creo una relacion de herencia Is que heredan de la clase JugadorCampo



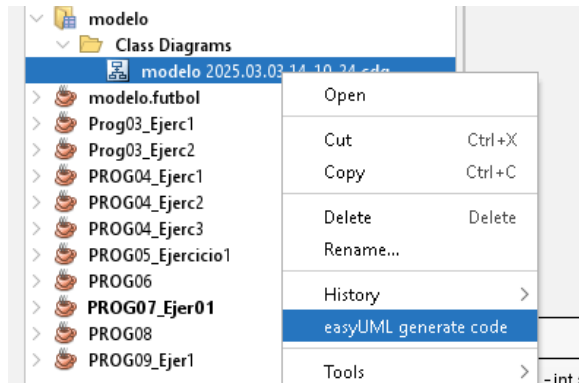
El diagrama completo quedaría de esta forma.



### Ejercicios 3

Genera el código Java correspondiente a la estructura UML producida en el punto anterior. ¿Cuál es la siguiente tarea a realizar a cargo del programador?

Para generar el código desde el diagrama de clases le doy clic derecho >> easyUML generate code y me genera el código automáticamente



Desde el diagrama de clases se genera el código, pero solo el esqueleto, lo siguiente que tendría que hacer el programador es:

Completar los constructores de cada clase para inicializar los atributos heredados y los atributos propios de cada clase.

Hay que implementar los métodos getter y setter correctamente para poder acceder y modificar los atributos.

Hay que definir el método mostrarInformacion para imprimir los datos de los jugadores de forma legible.

También hay que implementar el método main para que el programa se pueda ejecutar.