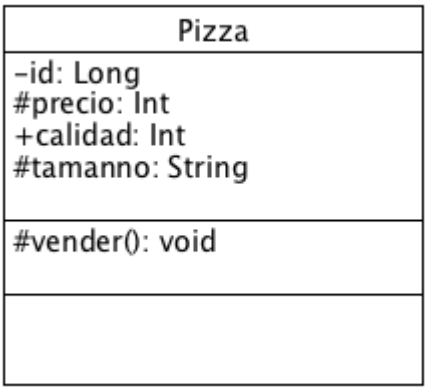
Ángel González M.

Solución Cuaderno de ejercicios: Diagramas de clase UML

Ejercicio 1: Pizza

Convierte el siguiente diagrama de clases de UML a código Java



Solución:

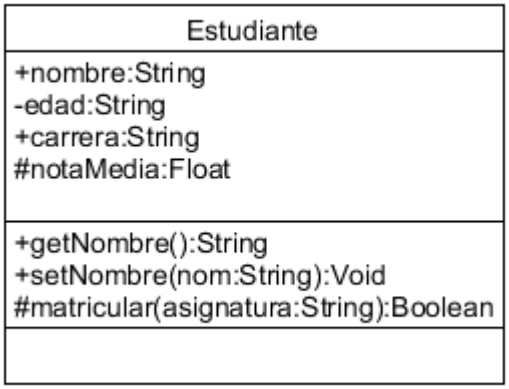
```
public class Pizza {
	private long id;
	protected int precio;
	public int calidad;
	protected String tamanno;

	protected void vender(){
		// TODO poner algo
	}

}
```

Ejercicio 2: Estudiante

Convierte el siguiente código escrito en Java a un diagrama de clases de UML

```
public class Estudiante {
	public String nombre;
	private String edad;
	public String carrera;
	protected float notaMedia;

	public String getNombre(){
		return nombre;
	}
	public void setNombre(String nom){
		this.nombre = nombre;
	}
	boolean matricular(String asignatura){
		System.out.println("Estudiante matriculado");
		return true;
	}

}
```

Solución



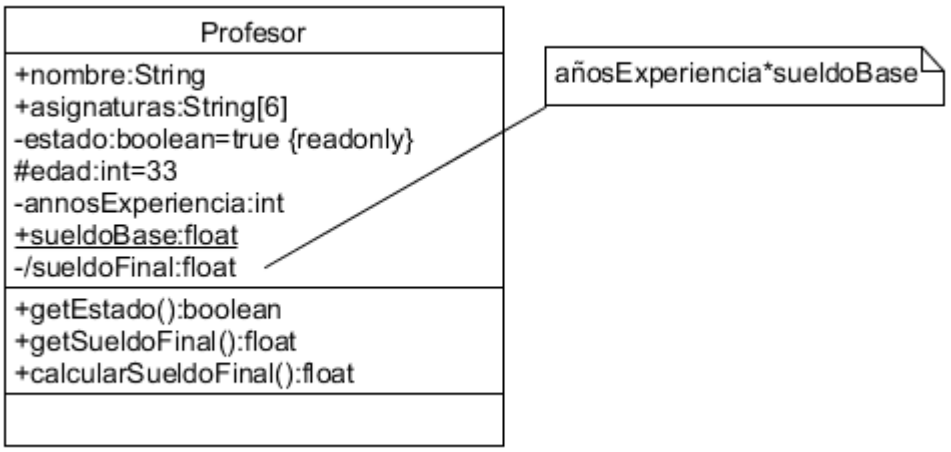
Ejercicio 3: Profesor

Convierte el siguiente código escrito en Java a un diagrama de clases de UML

```
public class Profesor {
	public String nombre;
	public String[] asignaturas = new String[6];
	private boolean estado = true;
	protected int edad = 33;
	private int annosExperiencia;
	public static float sueldoBase;
```

```
        private float sueldoFinal;

        public boolean getEstado(){
                return estado;
        }

        public float getSueldoFinal(){
                return calcularSueldoFinal();
        }

        private float calcularSueldoFinal(){
                return annosExperiencia * sueldoBase;
        }
}
```

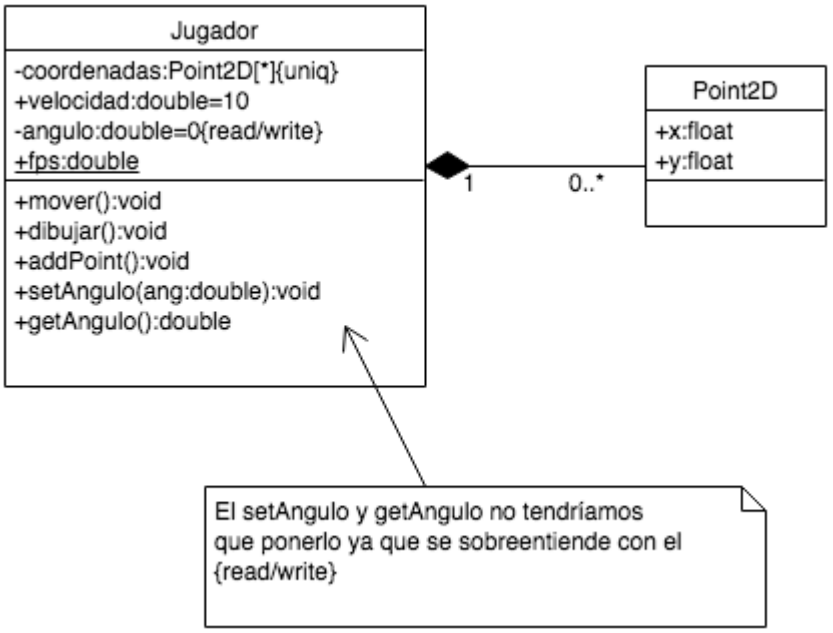
Solución



Ejercicio 4: Jugador

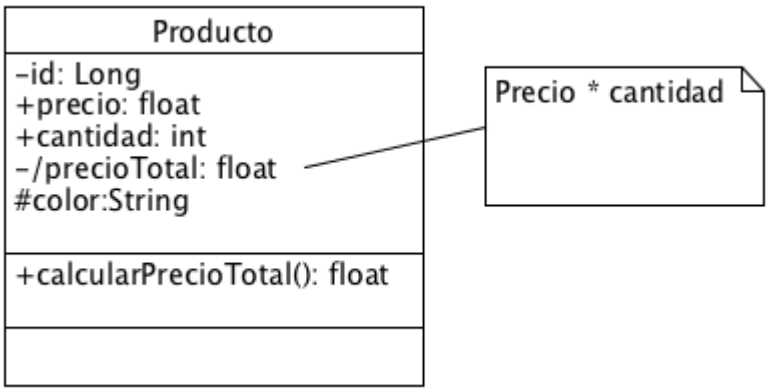
Convierte el siguiente código escrito en Java a un diagrama de clases de UML

```
import java.awt.geom.Point2D;
import java.util.HashSet;

public class Jugador {
	private HashSet<Point2D> coordenadas = new HashSet<Point2D>();
	public double velocidad = 10;
	private double angulo = 0;
	public static double fps;

	public void mover(){
		for(Point2D punto : coordenadas){
			double x = punto.getX() + (this.velocidad * Math.sin(this.angulo));
			double y = punto.getY() + (this.velocidad * Math.cos(this.angulo));
			punto.setLocation(x, y);
		}
	}

	public void dibujar(){
		for(Point2D punto : coordenadas){
			System.out.println(punto);
		}
	}

	public void setAngulo(double ang){
		this.angulo = ang;
		if(this.angulo <= 0){
			this.angulo = 0;
		}
		if(this.angulo >= 2*Math.PI){
			this.angulo = 2*Math.PI;
		}
	}

	public double getAngulo(){
		return this.angulo;
	}
}
```



Ejercicio 5: Producto

Convierte el siguiente diagrama de clases de UML a código Java



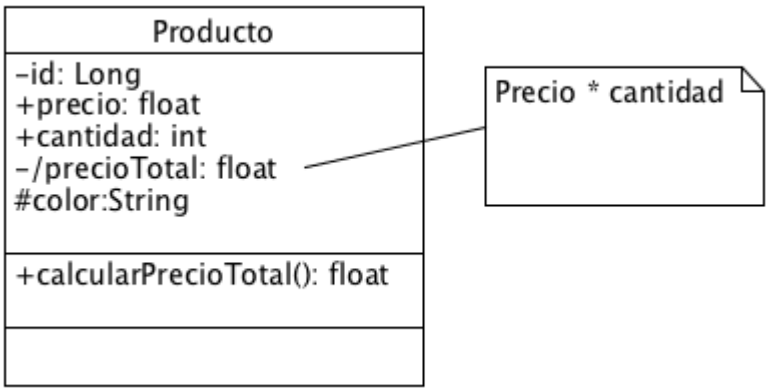
Solución:

```
package DIAGRAMAS_CLASE.Ejercicio5;

public class Producto {
	private long id;
	public float precio;
	public int cantidad;
	//private float precioTotal;
```

```
        protected String color;

        public float getPrecioTotal() {
                return precio*cantidad;
        }

        public float calcularPrecioTotal(){
                return precio*cantidad;
        }
}
```

Ejercicio 6: Artículo

Convierte el siguiente diagrama de clases de UML a código Java

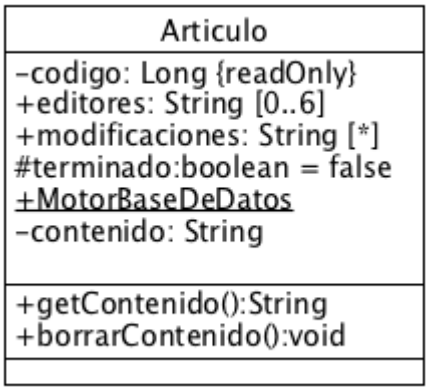



```
class Producto{
        private long id;
        private  float precio = 0;
        private int cantidad = 1;
        private float precioTotal;
        protected String color;

        private float calcularPrecioTotal(){
                precioTotal = precio*cantidad;
                return precioTotal;
        }
        public void setPrecio(float valor){
                precio = valor;
                calcularPrecioTotal();
```

```
        }
        public void setCantidad(float valor){
                cantidad = (int)valor;
                calcularPrecioTotal();
        }
}

public class Principal {
        public static void main(String[] args) {
                Producto lapiz = new Producto();
                //lapiz.precio = 22;
                //lapiz.cantidad= 3;
                //lapiz.calcularPrecioTotal();
                lapiz.setPrecio(22);
                lapiz.setCantidad(3);
        }
}
```

Ejercicio 6: Reservas

Convierte el siguiente diagrama de clases de UML a código Java

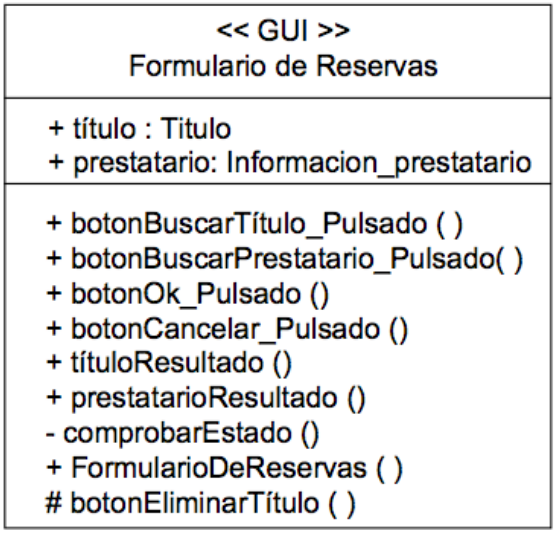



```
class Articulo{
        private long codigo;
        public String[] editores = new String[6];
        public String[] modificaciones;
        protected boolean terminado = false;
        public static String motorBaseDeDatos;
        private String contenido;
```

```
	public String getContenido(){
		return contenido;
	}
	public void borrarContenido(){
		contenido = "";
	}
	public long getCodigo(){
		return codigo;
	}
}


public class Principal {

	public static void main(String[] args) {
		Articulo art1 = new Articulo();
		art1.terminado = false;
		Articulo art2 = new Articulo();
		art2.terminado = true;
		Articulo art3 = new Articulo();
		art3.terminado = true;

		art1.motorBaseDeDatos = "MySQL";
		System.out.println(art3.motorBaseDeDatos); // MySQL
		System.out.println(art2.motorBaseDeDatos); // MySQL
		System.out.println(art1.motorBaseDeDatos); // MySQL

	}

}
```

Ejercicio 7: Reservas

Convierte el siguiente diagrama de clases de UML a código Java



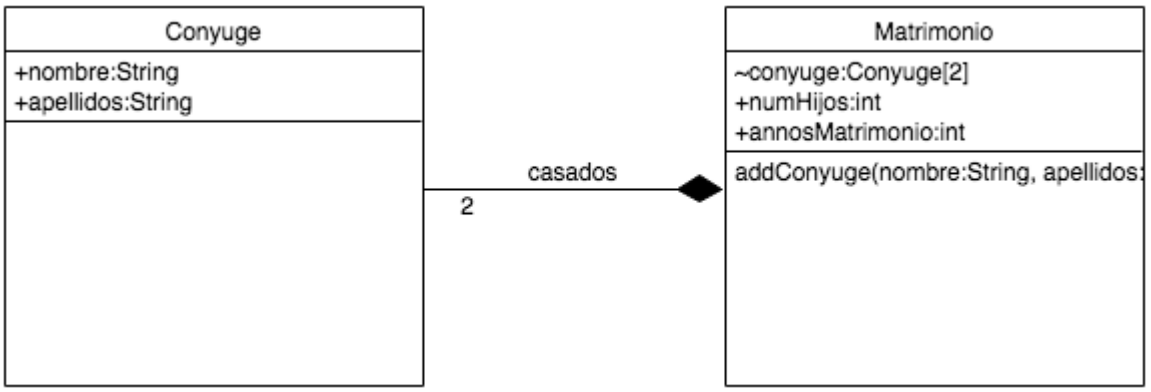

```
class Titulo{

}
class InformacionPrestatario{

}
class FormularioReservas{
        public Titulo titulo;
        public InformacionPrestatario prestatario;

        public void botonBuscarTituloPulsado(){

        }
        public void botonPulsarPrestatarioPulsado(){

        }
        public void botonOKPulsado(){

        }
        public void botonCancelarPulsado(){

        }
        public void tituloResultado(){

        }
        private void comprobarEstado(){

        }
        public void formularioDeReservas(){

        }
        protected void botonEliminarTitulo(){
```

```
      }
}
```

Ejercicio 8: Matrimonio

Convierte el siguiente código escrito en Java a un diagrama de clases de UML Sustituye los … por lo que consideres oportuno.

```
class Conyuge {
   ...
}
class Matrimonio {
   ...
   Conyuge[] conyuges = new Conyuge[2];
   ...
   Matrimonio() {
      conyuges[0] = new Conyuge();
      conyuges[1] = new Conyuge();
      ...
   }

   addConyuge(String nombre){
     Conyuge c = new Conyuge(nombre);
     conyuges.add(c)
   }
}
```



Ejercicio 9: Empresa/cliente

Convierte el siguiente diagrama de clases de UML a código Java

```
import java.util.ArrayList;

class Cliente{
	private String nombre;
	private String telefono;


}
class Empresa{
	private ArrayList<Cliente> clientes = new ArrayList<Cliente>();

	public void addCliente(Cliente cliente){
		clientes.add(cliente);
	}
}
```

Ejercicio 10: Employee

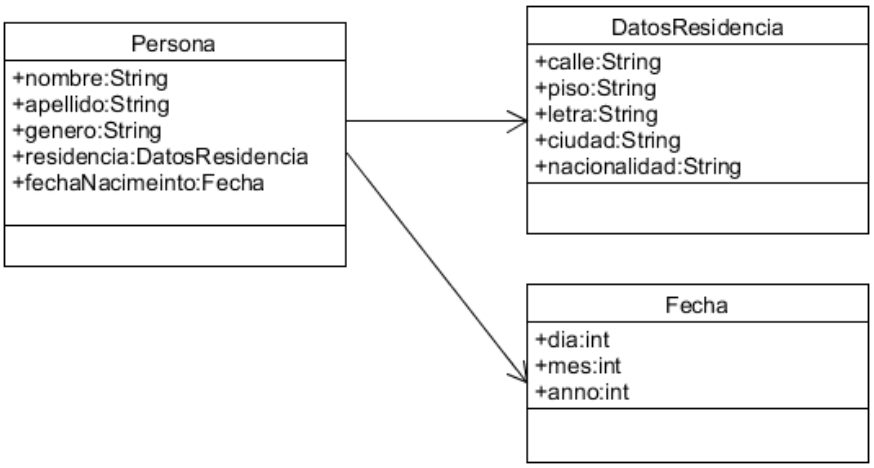
public class Employee {

```
    private String name;
    private double payRate;
    private final int EMPLOYEE_ID;

    private static int nextID = 1000;

    public static final double STARTING_PAY_RATE = 7.75;

    public Employee(String name) {
        this.name = name;
        EMPLOYEE_ID = getNextID();
        payRate = STARTING_PAY_RATE;
    }

    public Employee(String name, double startingPay) {
        this.name = name;
        EMPLOYEE_ID = getNextID();
        payRate = startingPay;
    }

    public String getName() { return name; }

    public int getEmployeeID() { return EMPLOYEE_ID; }

    public double getPayRate() { return payRate; }

    public void changeName(String newName) { name = newName; }

    public void changePayRate(double newRate) { payRate = newRate; }

    public static int getNextID() {
        int id = nextID;
        nextID++;
        return id;
    }
}
```

Ejercicio 11: Información personal

Se necesita almacenar (usa clases) la información de una persona. En concreto su nombre, apellidos, lugar de residencia (calle, piso, letra, cp, ciudad, nacionalidad), su género y su fecha de nacimiento (en concreto guardar por separado su día, mes y año).

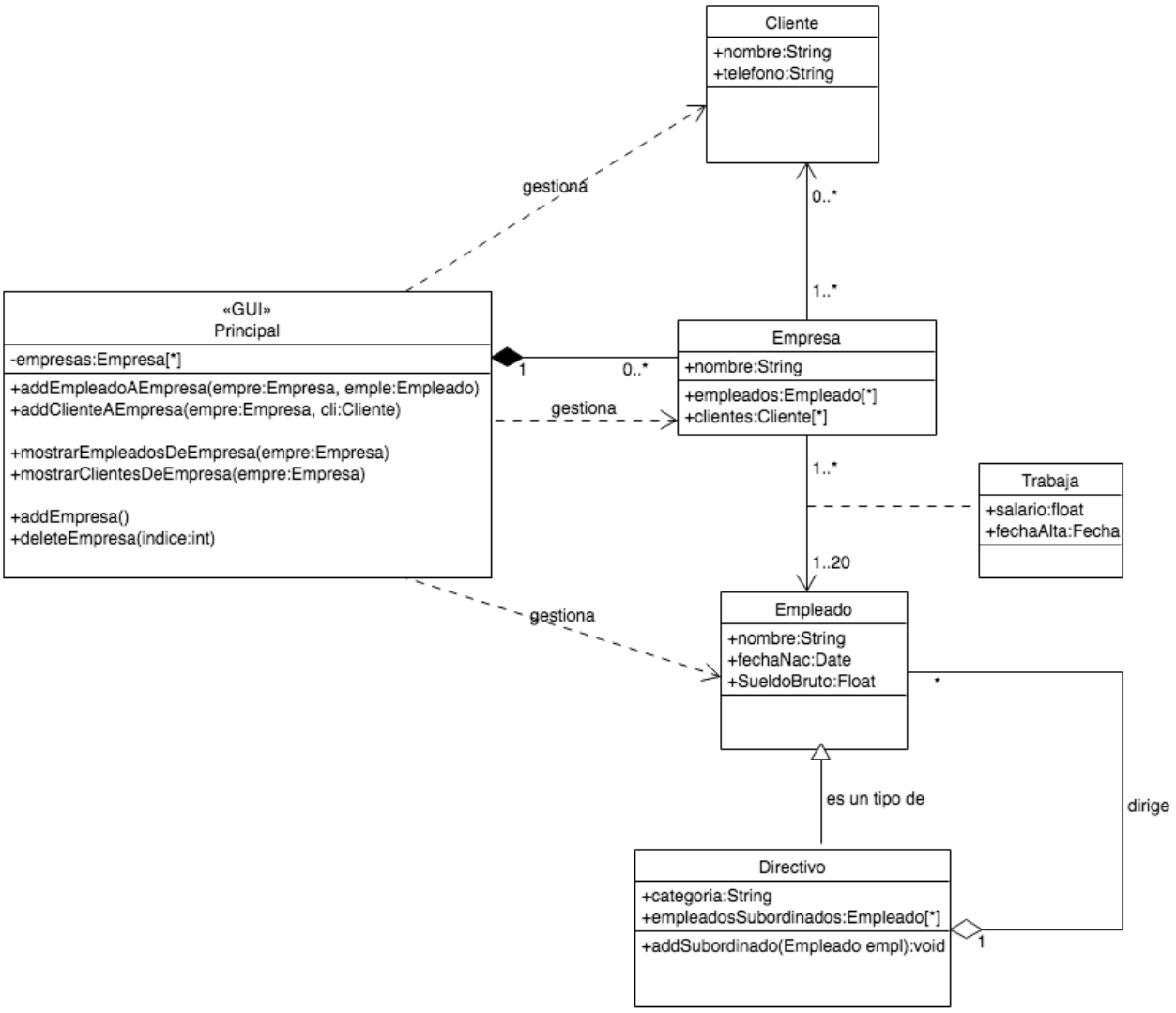
Crea el diagrama de clases UML correspondiente.



Ejercicio 12: Empresas

Representa mediante un diagrama de clases la siguiente especificación:

- Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes.
- Ambos(empleados y clientes) se caracterizan por su nombre y edad.
- Los empleados tienen un sueldo bruto. Los empleados que son directivos tienen una categoria, asi como un conjunto de empleados subordinados.
- De los clientes además se necesita conocer su teléfono de contacto.
- La aplicación necesita mostrar los datos de empleados y clientes.
- La empresa necesita conocer el salario y la fecha de alta de cada uno de sus empleados.



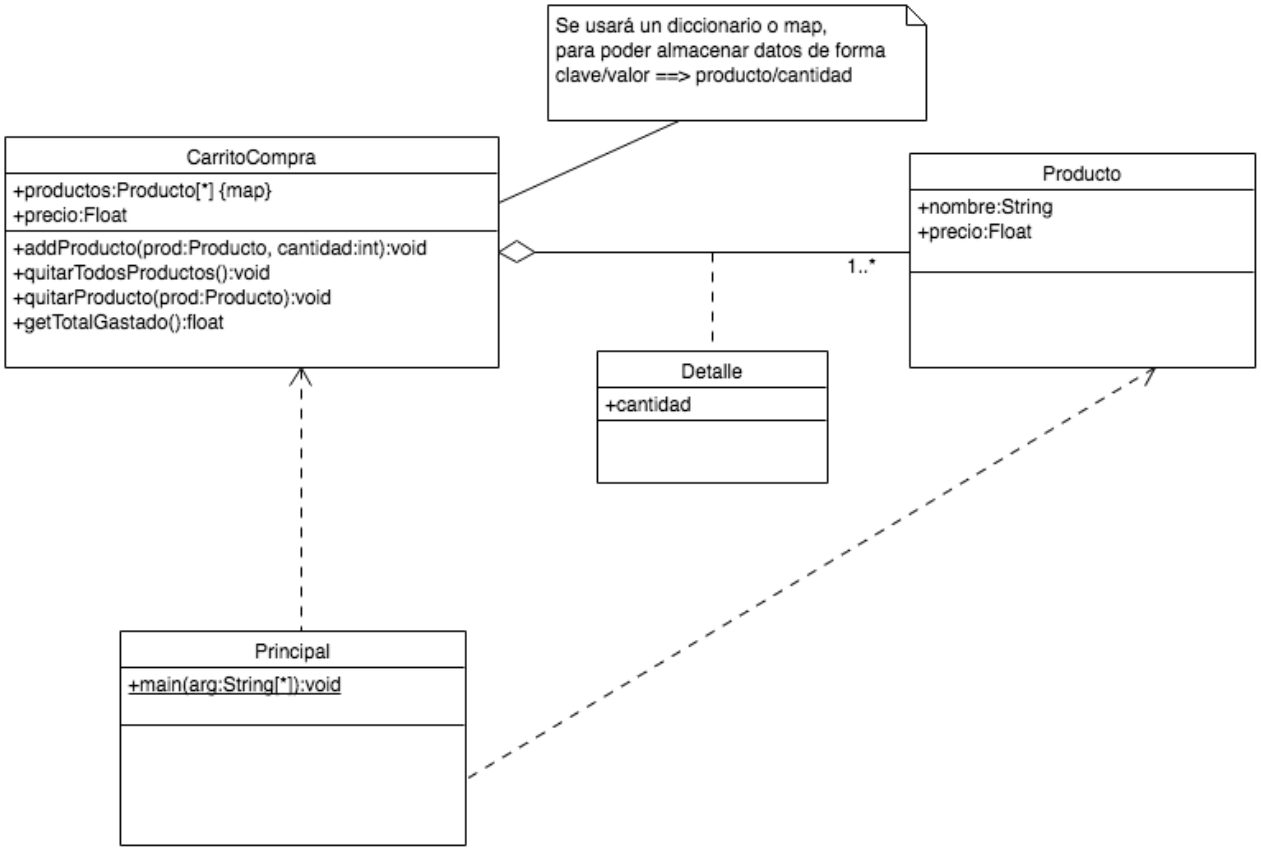
Ejercicio 13: Ejercicio personal

Ejercicio personal

Ejercicio 14: Carrito compra

Crea un programa que pida al usuario datos de productos que va comprando
{nombre, precio, cantidad}
Estos productos los irá almacenando en una colección (al estilo de un carrito de la compra online)
Cuando termine de insertar productos saldrá por pantalla el total gastado

- Crea el diagrama de clases
- Haz un program en java



Ejercicio 15: Pedidos

Crea el diagrama de clases de la siguiente aplicación:

Tenemos un sistema donde los clientes realizan pedidos de productos y los pueden pagar.

Un cliente podrá realizar múltiples pedidos, pero un pedido solo podrá estar asociado a un cliente.
Los datos que se necesitan guardar de los clientes son nombre, dirección, email y si está activo o no en el sistema.

De los pedidos es necesario almacenar el código y la fecha de creación del pedido, así como 2 métodos: uno de validación de los datos del pedido y otro para proceder al envío del pedido.

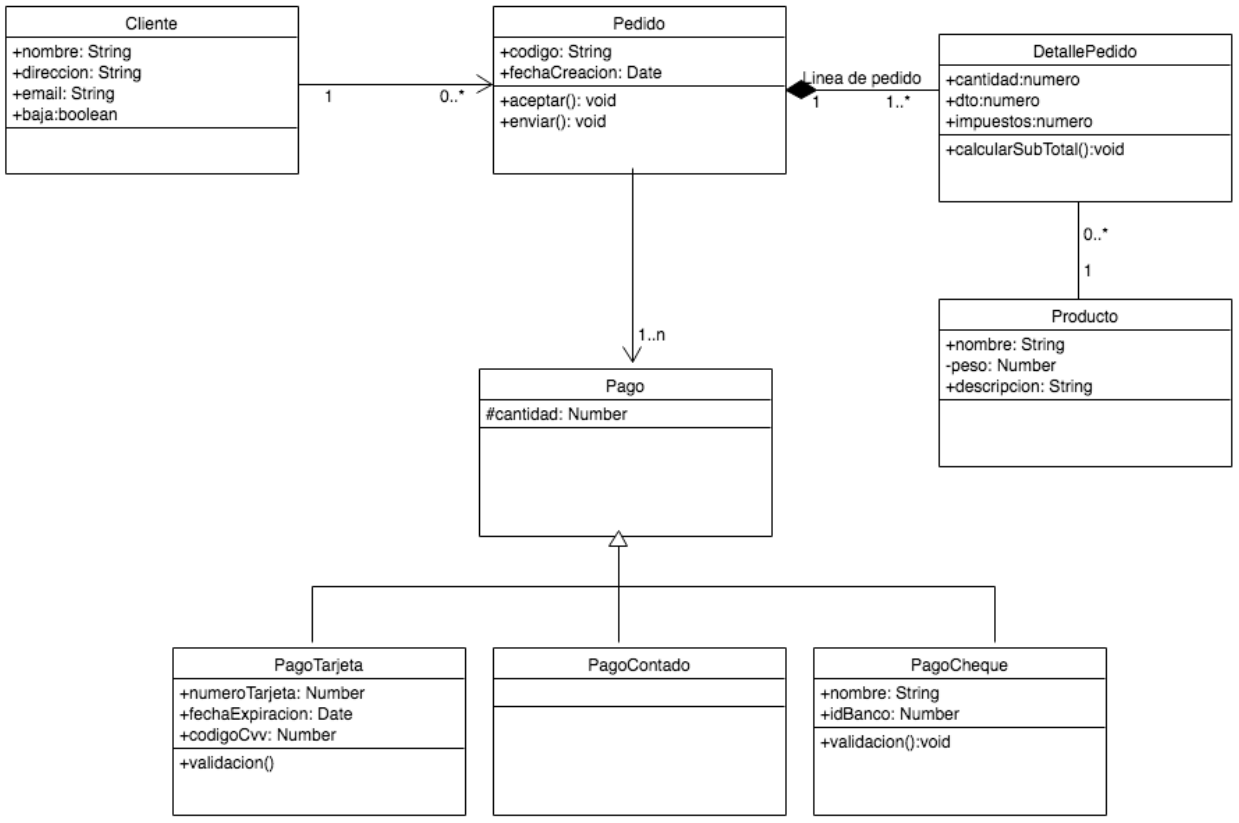
El sistema también tendrá productos de los cuales necesitaremos almacenar su nombre, el peso y una descripción.

Con lo cual será necesario asociar los pedidos a esos productos. Eso lo hacemos con los detalles de pedido.
Un pedido tendrá múltiples detalles de pedido. Un detalle de pedido estará asociado a un producto en particular.
En los detalles de pedido vamos a almacenar la cantidad, los descuentos y los impuestos aplicables a ese producto.

Los pedidos estarán asociados a uno o múltiples métodos de pago.
La aplicación trabaja con 3 tipos de pagos:

- pagos con tarjeta: número de tarjeta, fecha aspiración y codigoCvv
- pago en metálico
- pago con cheque: el nombre del pagador y el id del banco

Los pagos tendrán que almacenar la cantidad del pago; y los pagos con tarjeta y con cheque tendrán un método que sirve para validar la cuenta.

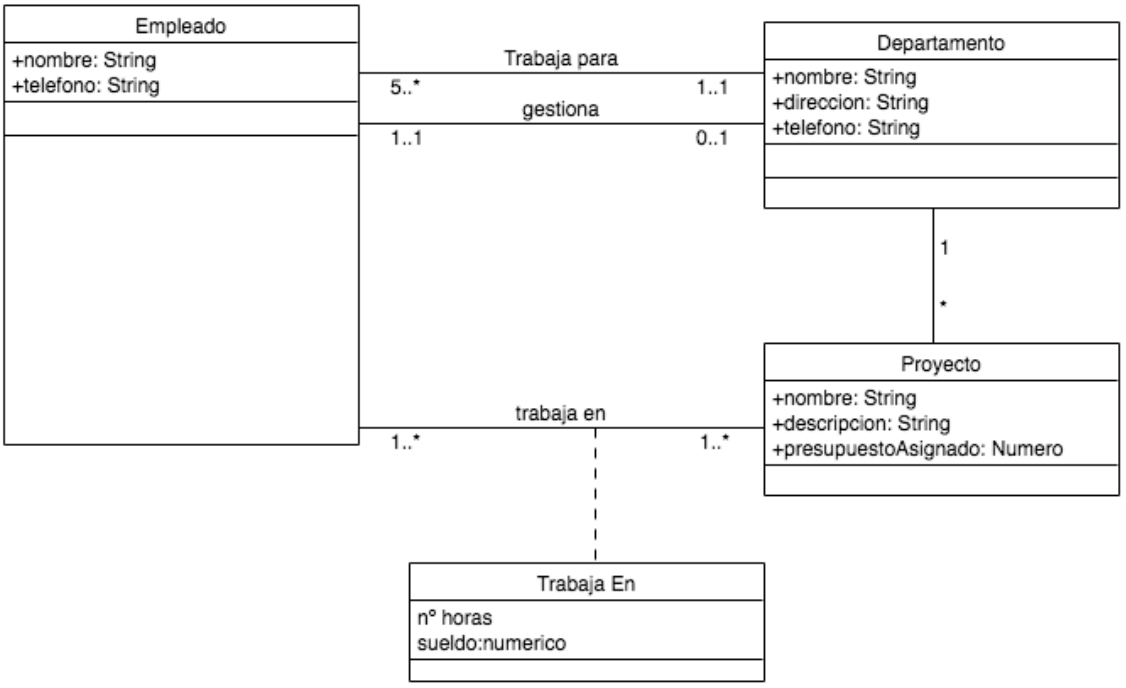


Ejercicio 16: Departamentos

Crea el diagrama de clases de la siguiente aplicación:

Tenemos un sistema donde nuestros trabajadores trabajan en departamentos.
Un trabajador solo puede trabajar en un departamento, y en un departamento hay como mínimo 5 trabajadores.
Cada departamento gestiona varios proyectos, pero un proyecto solo podrá ser llevado por un departamento.
Un empleado será el encargado de gestionar (jefe) un departamento.
Además los empleados podrán trabajar en distintos proyectos, y por su puesto en un proyecto estarán trabajando múltiples empleados.

La información que hay que almacenar de los empleados será: el nombre y el teléfono
La información que hay que almacenar de los departamentos será: el nombre, la dirección y el teléfono
La información que hay que almacenar de los proyectos será: el nombre, una descripción y el presupuesto económico asignado
También será necesario guardar el número de horas y el sueldo de cada trabajador en cada proyecto.



Ejercicio 17: Artículos

Crea el diagrama de clases de la siguiente aplicación:
El sistema permitirá que usuarios suban artículos creados por ellos.

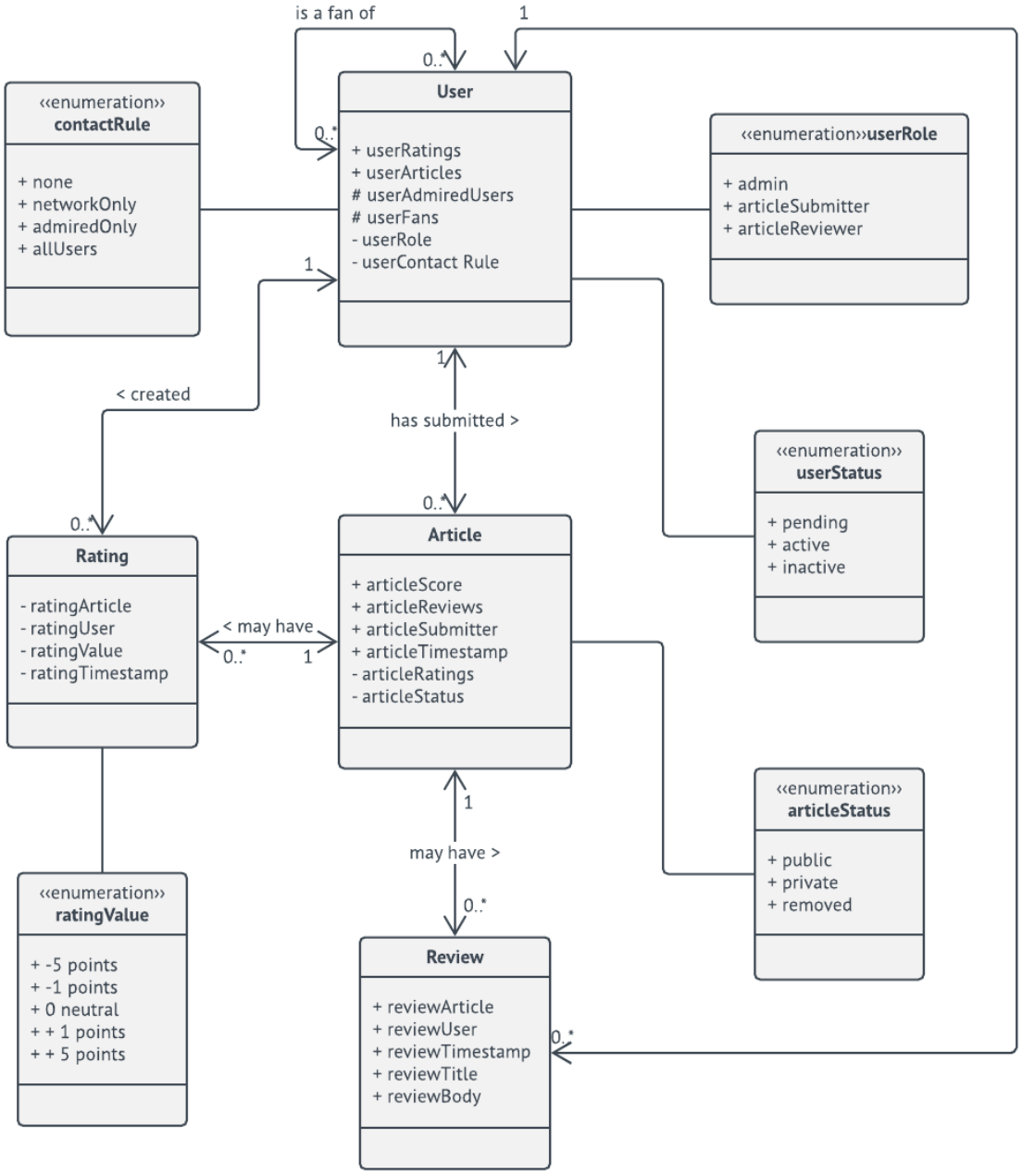
- Los usuarios podrán puntuar(rating) otros artículos (de -5 a 5 puntos) esto se modelará con una clase llamada Rating. La puntuación asignada se modelará como una enumeración con los valores del -5 al 5.
- Los usuarios tendrán un rol. Este rol se modelará como una enumeración con 3 valores (admin, articleSubmiter o articleReviewer)
- Los usuarios también tendrán un estado, que se modelará con una enumeración con los valores: pending, active e inactive.

Un usuario podrá hacerse fan de otros usuarios
Los artículos tendrán un estado, este se modelará como una enumeración con los valores public, private y removed.

Y los usuarios podrán revisar artículos, esto se modelará con la clase Review

- Una review consistirá de un título y un cuerpo (descripción).

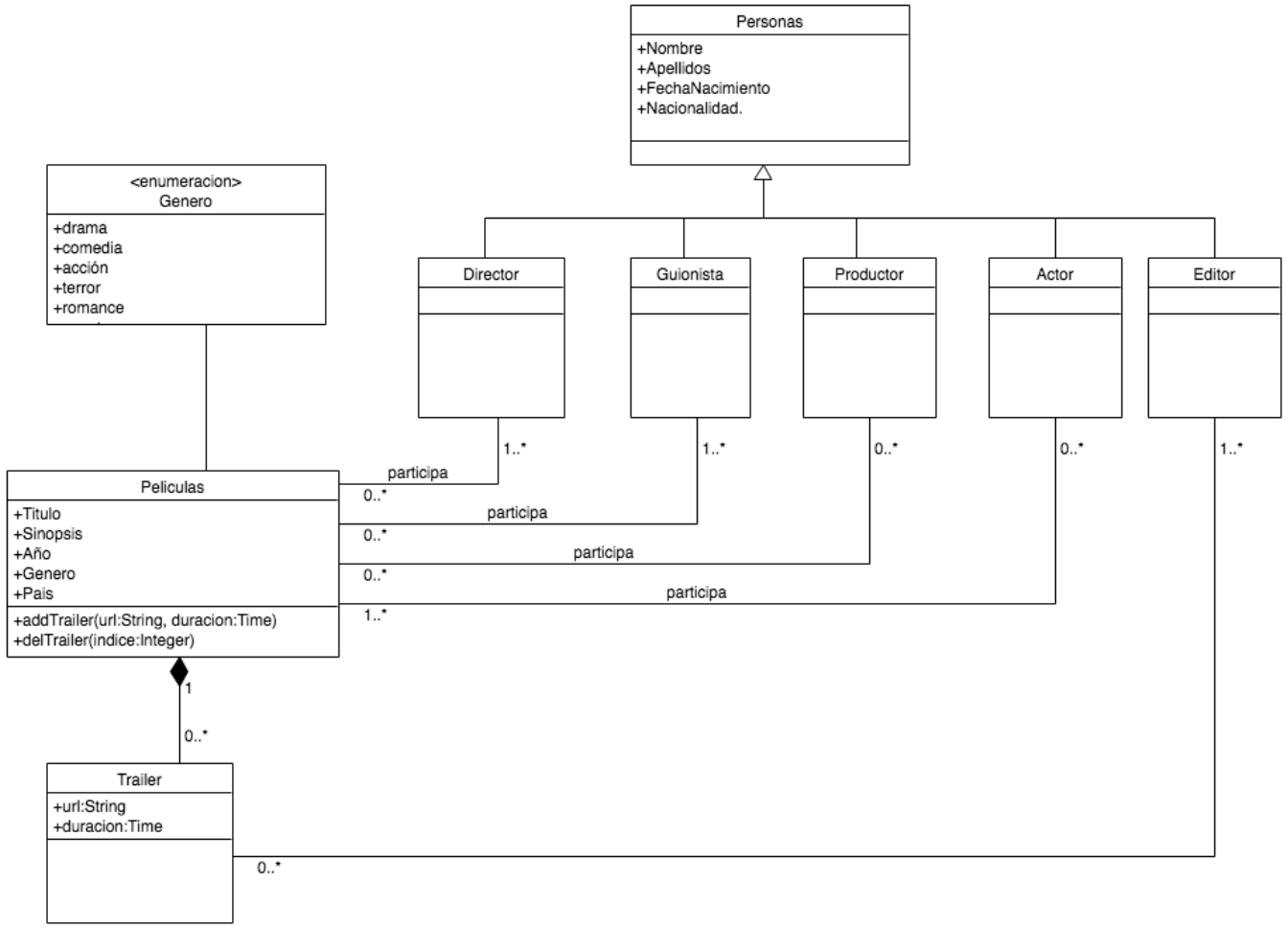
Ángel González M.



Ejercicio 18: Película

Representa mediante un diagrama de **clases** la siguiente especificación sobre las personas que participan en una película

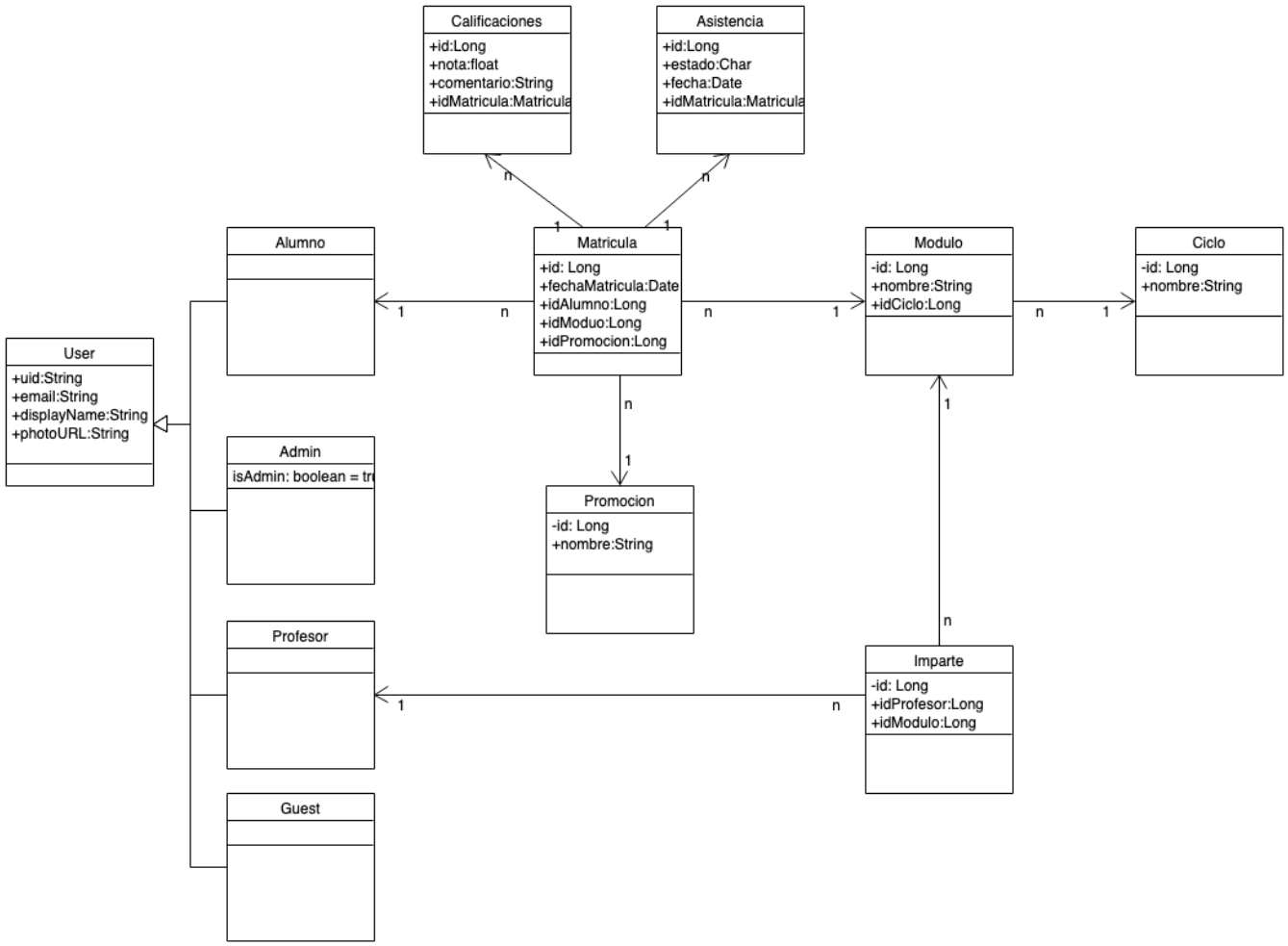
- De cada película se almacena el título, la sinopsis, el año, el género al que pertenece (drama, comedia, acción, terror, romance, aventura, scifi) y el país.
- Sobre las personas que participan en la película es necesario conocer el nombre, los apellidos, la fecha de nacimiento y la nacionalidad.
- Una persona puede participar en una película como actor, director, productor o guionista.
- Una película tiene al menos un director y un guionista.
- Una persona se considera actor si ha actuado al menos en una película.
- Una película puede tener asociados varios trailers que son editados por una o más personas. No puede existir el trailer de una película hasta que existe la película.
- Los trailers tiene un atributo url de tipo texto y duración de tipo Time



Ejercicio 19: Cebem

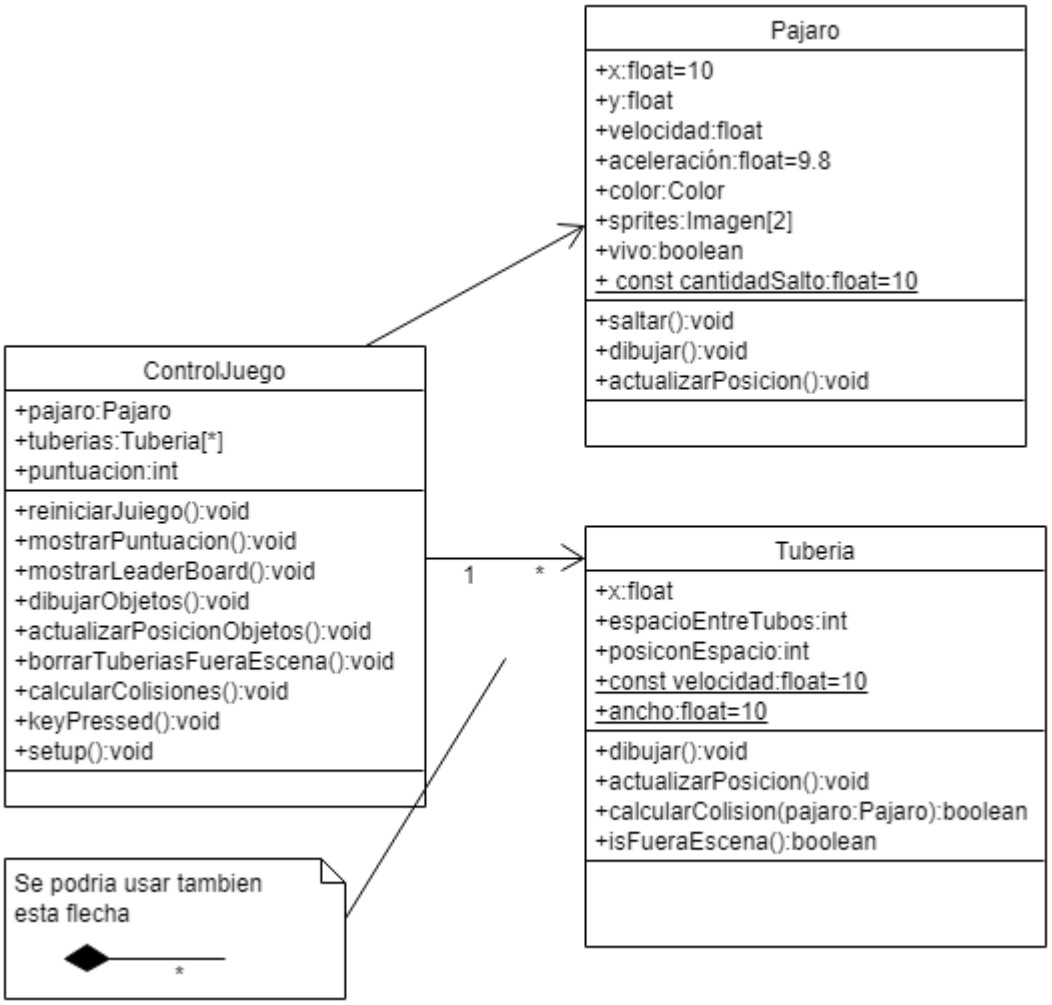
Sistema de gestión del centro formación Cebem

Construye el diagrama de clases UML del ejercicio Ejercicio 4
Solución:



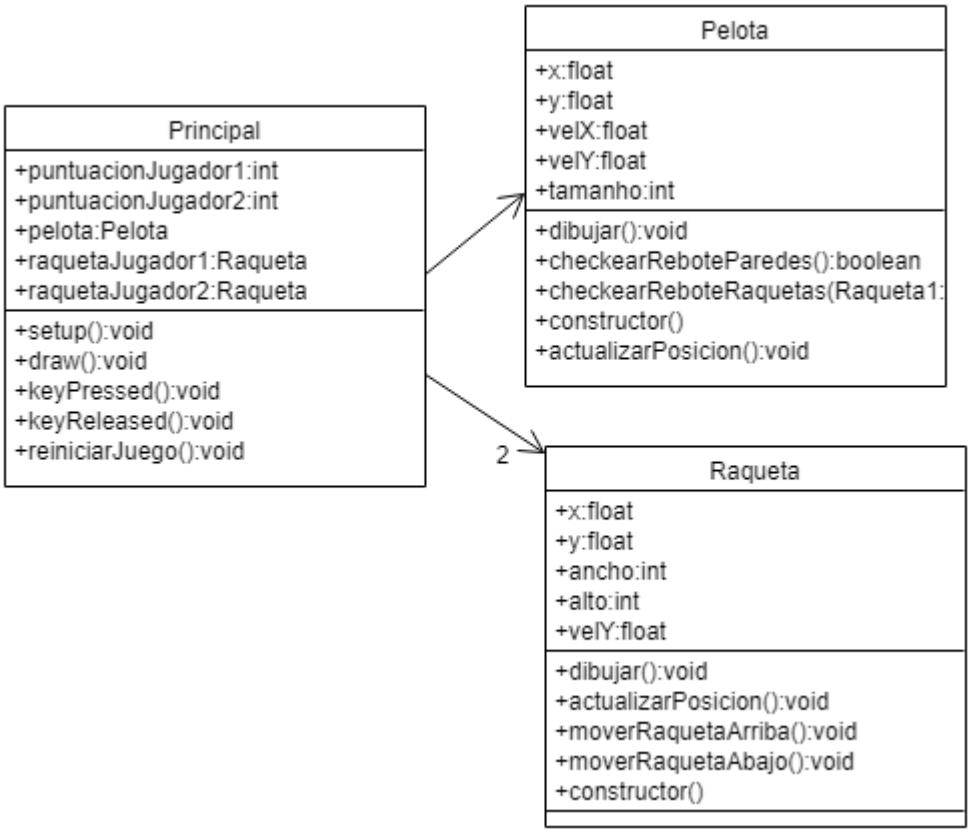
Ejercicio 20: Juego FlappyBird

Construye el diagrama de clases UML del ejercicio



Ejercicio 21: Juego Pong

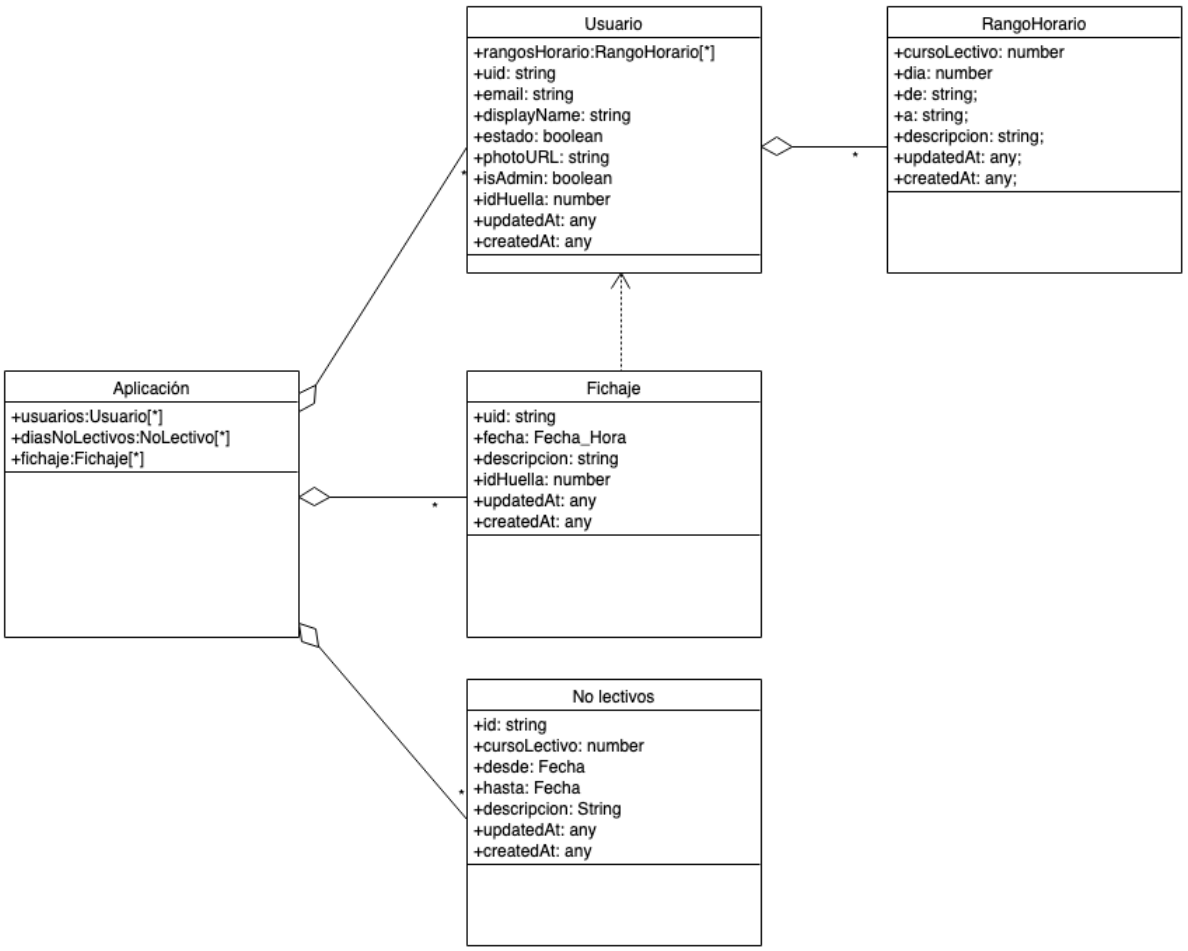
Construye el diagrama de clases UML del ejercicio Ejercicio 7
Solución



Ejercicio 22: Fichador (diagrama de clases)

Partiendo del ejercicio 8 se te pide que desarrolles el diagrama de clases del proyecto.

Solución



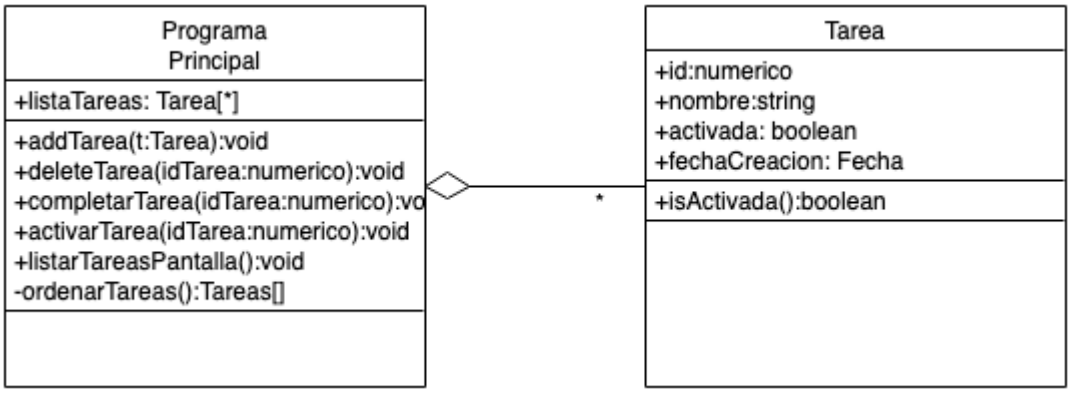
Ejercicio 23: Préstamo de libros de clase (diagrama de clases)

Partiendo del ejercicio 9 se te pide que desarrolles el **diagrama de clases** del proyecto.

Ejercicio 24: Tareas

Partiendo del ejercicio 10 se te pide:

- Que desarrolles el **diagrama de clases** del proyecto de gestión de tareas.
- Así como su **implementación** en Java



Ejercicio 25: Doom

En 1993 John Carmack junto a John Romero crean el videojuego **DOOM** cuando trabajaban para la empresa estadounidense *Id Software*. Se pide que crees el **diagrama de clases** (simplificado) **en UML** de este magnifico juego.
Nota: vamos a usar los nombre originales en inglés.

En el juego de DOOM los elementos del juego se guardan en una clase principal que ellos llaman **Lump**. Esta clase tendrá los siguientes elementos:

- Una lista de *paletas*, en concreto 14, usado para guardar los conjuntos de colores del juego (en aquellos años el motor sólo podía mostrar un máximo de 256 colores simultáneos)
- Una lista de *things*, usado para guardar elementos que nos encontramos en el escenario: enemigos, armas, llaves, pociones, etc
- Una lista de *lineDefs* que servirán para definir las paredes de nuestro mapa
- Además esta clase *Lump* tendrá 2 métodos:
    - uno para dibujar el mapa en pantalla
    - y otro para cargar los datos del mapa del disco duro.

De la clase **Paleta (Palete)** se guardarán en 3 atributos numéricos:

- la componente de color rojo (r),
- la componente verde (g)
- y la componente azul del color (b).
- Además esta clase (*palete)* tendrá 2 métodos uno para obtener el color en RGB y otro para obtener el color en hexadecimal

De la clase **Things** es necesario guardar:

- la posición *x*, y la posición *y*,
- así como el ángulo (angle) en la que se posicionará ese objeto en el mapa.
- Por su puesto también será necesario guardar en esta clase (mediante un entero) el tipo(*type*) de objeto que se quiere representar.
- También tendrá un método para *mover* el objeto a un coordenada x e y

- y otro método para *rotar* r grados el objeto

En cuanto a la clase **LineDef** (que nos representa las paredes del mapa) tendremos que guardar los siguiente elementos:

- Dos vértices *(v1* y *v2*) que delimitan la posición de esa pared en el juego
- Dos sideDefs (*sideRight* y *sideLeft*) que indica cómo se deberá pintar (mapear) la superficie de esa pared (son necesarios dos porque las paredes tienen 2 lados)

En cuanto a los **Vértices** (también será una clase) se deberá guardar su coordenada *x* y su coordenada *y*

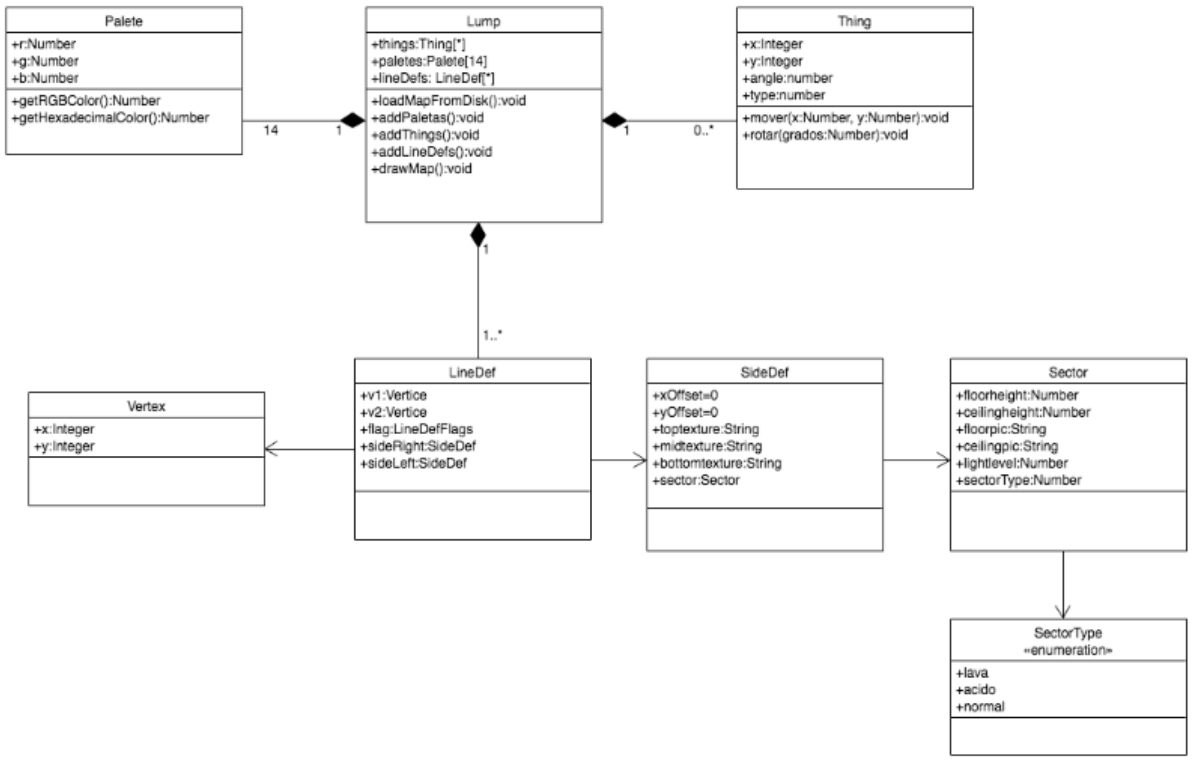
En cuanto a la clase **SideDef** será necesario guardar la siguiente información:

- La ruta de la textura superior (tipo texto) (*toptexture*)
- La ruta de la textura media (tipo texto) (*midtexture*)
- La ruta de la textura inferior(tipo texto) (*bottomtexture*)
- Y el *sector* en el cual se encuentra este sideDef. Este sector nos servirá para representar una habitación completa.
- El desplazamiento horizontal o vertical de la textura (*xOffset* e *yOffset*) que por defecto tendrán valor cero.

Para la clase **Sector** tendremos que guardar:

- la altura del suelo de la habitación (*floorheight*)
- la altura del techo de la habitación (*ceilingheight*)
- La ruta de la textura (tipo texto) para el suelo (*floorpic*)
- La ruta de la textura (tipo texto) para el techo (*ceilingpic*)
- El nivel de luz de este sector (valor numérico) (*lightlevel*)
- Un flag (de tipo enumeración numérica) que nos indica qué tipo de suelo tiene este sector (lava, ácido, normal, etc) (*sectorType*)

Solución:



Ejercicio 26: Jugadores

Se necesita almacenar (usa clases) la información un jugador en un videojuego.
En concreto del **jugador** necesitamos almacenar su *posicionXPantalla*, *posicionYPantalla* y *numeroVidas*.
Además el jugador poseerá los métodos u operacinoes *mover*, *girar* y *disparar*.
Todos estos atributos y métodos serán públicos excepto *numeroVidas* que será privado.

El jugador también tendrá una *lista* (ArrayList) con los enemigos abatidos (otra clase nueva).

Existirán 2 tipos de jugadores los **jugadores registrados** y los **jugadores invitados**.

En concreto los jugadores registrados tendran 2 atributos más: su *nombre* y los datos de su fecha de nacimiento (en concreto guardar por separado su día, mes y año) estos datos tendrán que estar guardados en una clase a parte llamada **FechaNac**.

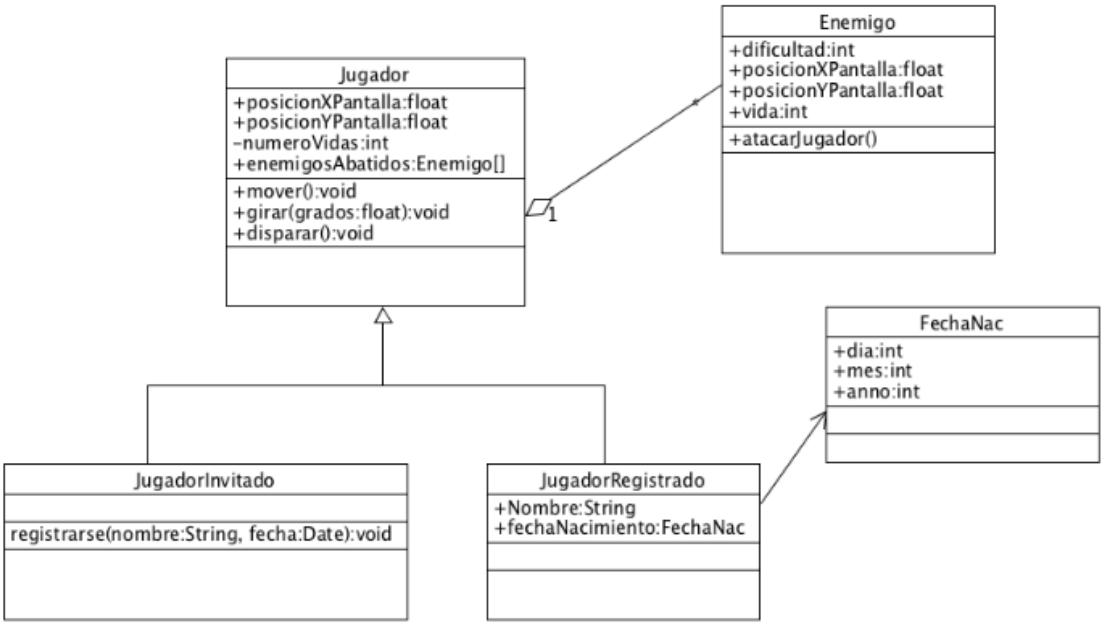
A su vez, los jugadores invitados tendrán un método u operación llamada *registrarse*(nombre:String, fechaNac:Date):void

Los **enemigos** tendrán como atributos: *dificultad*, *posicionXPantalla*, *posicionYPantalla* y *vida* y como métodos u operaciones *mover*, *girar* y disparar.

El programa en total tendrá que tener 5 clases.

Se pide: Crea el **diagrama de clases UML** correspondiente con sus relaciones adecuadas.

Solución



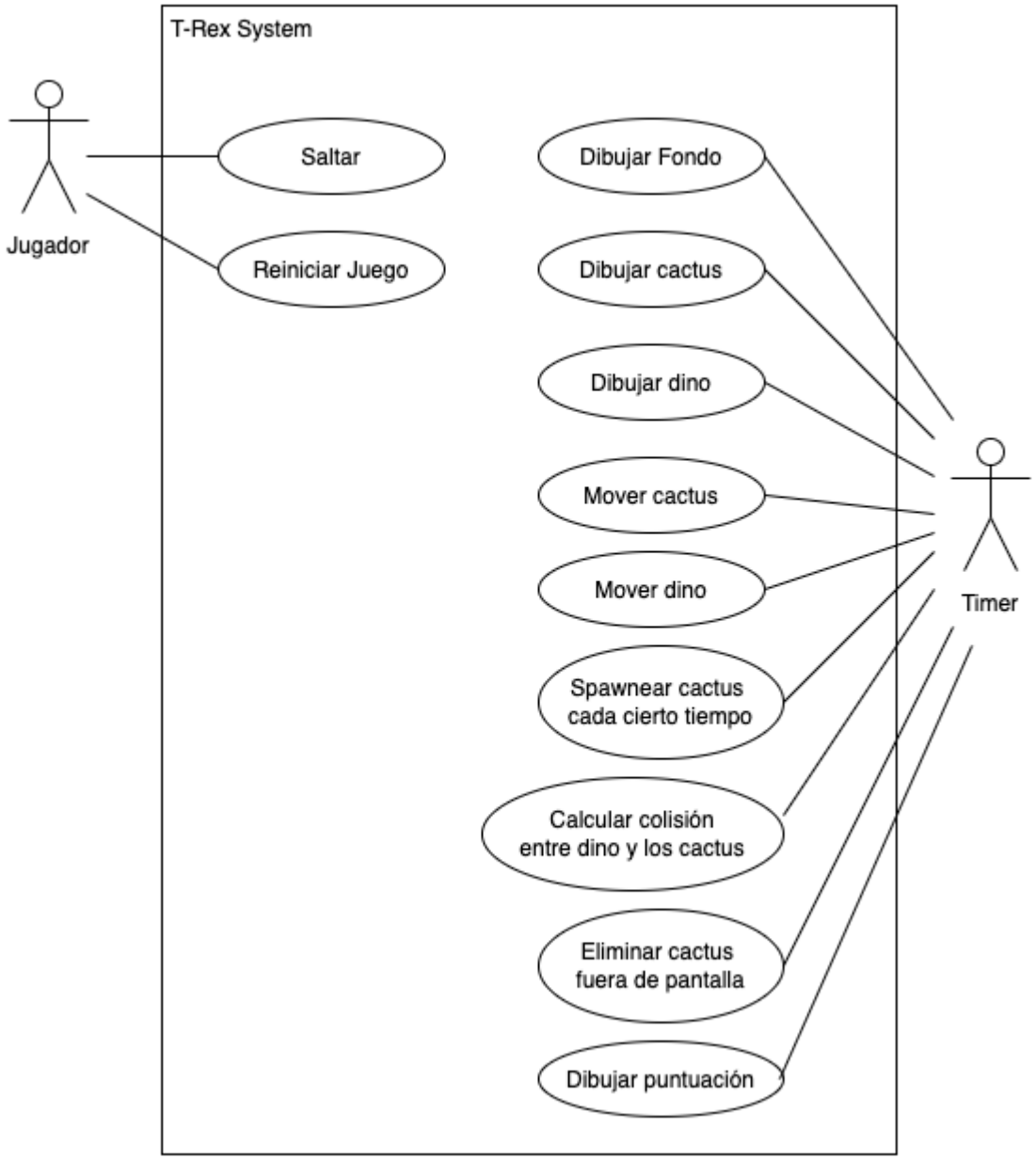
Ejercicio 27: TRex Dino Chrome

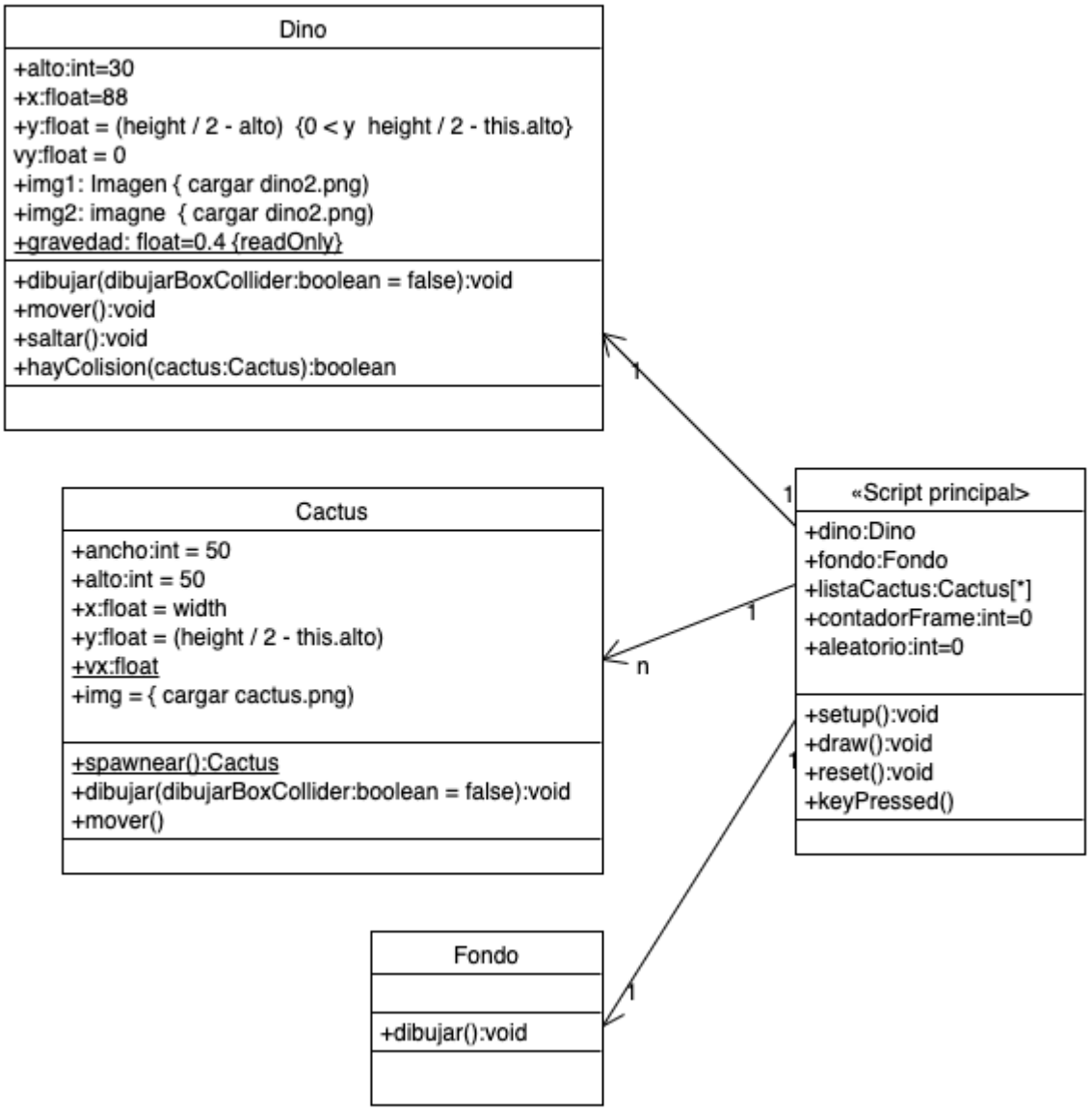
Crea el diagrama:

- El diagrama de casos de uso
- El diagrama de clases
- El diagrama de actividad del bucle principal

del juego dinosarurio T-Rex de Chrome chrome://dino/

Ángel González M.




Dino
+alto:int=30
+x:float=88
+y:float = (height / 2 - alto) {0 < y height / 2 - this.alto}
vy:float = 0
+img1: Imagen { cargar dino2.png)
+img2: imagne { cargar dino2.png)
+gravedad: float=0.4 {readOnly}
+dibujar(dibujarBoxCollider:boolean = false):void
+mover():void
+saltar():void
+hayColision(cactus:Cactus):boolean
Cactus
+ancho:int = 50
+alto:int = 50
+x:float = width
+y:float = (height / 2 - this.alto)
+vx:float
+img = { cargar cactus.png)
+spawnear():Cactus
+dibujar(dibujarBoxCollider:boolean = false):void
+mover()
«Script principal>
+dino:Dino
+fondo:Fondo
+listaCactus:Cactus[*]
+contadorFrame:int=0
+aleatorio:int=0
+setup():void
+draw():void
+reset():void
+keyPressed()
Fondo
+dibujar():void
1
1
1
n
1
1

Ángel González M.