Lucas Varela Negro Dam1

Tema 5: Introducción a UML y la Orientación a objetos

Práctica 5.1: Introducción de elementos de la programación orientada a objetos en código fuente:

Busca en internet trozos de código en distintos lenguajes e identifica los distintos elementos de la programación orientada a objetos: constructores, herencia, polimorfismo, sobrecarga de métodos, asociaciones…

C++:

- Constructores:

```
class MyClass {     // The class
  public:           // Access specifier
    MyClass() {     // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;    // Create an object of MyClass (this will call the constructor)
  return 0;
}
```

Podemos ver en el ejemplo de la imagen: la clase, el tipo de acceso, el constructor y el objeto que llama al constructor.

- Herencia:

```
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
    }
};

// Derived class
class Car: public Vehicle {
  public:
    string model = "Mustang";
};

int main() {
  Car myCar;
  myCar.honk();
  cout << myCar.brand + " " + myCar.model;
  return 0;
}
```

Como se puede ver en el ejemplo, se crea una clase Vehículo con unas características y métodos que luego son heredadas por una nueva clase llamada coche mediante la sintaxis siguiente: ***class*** *coche:* ***public*** *vehículo {*

- Polimorfismo:

```
// Base class
class Animal {
  public:
    void animalSound() {
    cout << "The animal makes a sound \n" ;
  }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
    cout << "The pig says: wee wee \n" ;
  }
};

// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
    cout << "The dog says: bow wow \n" ;
  }
};
```

En el ejemplo se muestra como se crea una clase con unos métodos que luego pueden ser utilizados por diferentes clases de diferentes maneras, siendo **la sintaxis** igual que en la herencia.

- Sobrecarga de métodos:

```
#include <iostream>
using namespace std;
class addition
{
public:
int addMethod(int x, int y)
{
return x + y;
}
int addMethod(int x, int y, int z)
{
return x + y + z;
}
};
int main(void)
{
addition add;
cout << add.addMethod(2, 3) << endl;
cout << add.addMethod(2, 3, 6) << endl;
return 0;
}
```

En el ejemplo se muestran dos métodos a los que se les pasa a uno 2 valores y al otro 3 valores, en el primero se añadirán 2 valores mostrando el resultado de su operación en el tercero 3 se añadirán 3 valores mostrando también el resultado de su operación.

- Asociaciones:

```cpp
#include <functional> // reference_wrapper
#include <iostream>
#include <string>
#include <vector>

// Since Doctor and Patient have a circular dependency, we're going to forward declare Patient
class Patient;

class Doctor
{
private:
    std::string m_name{};
    std::vector<std::reference_wrapper<const Patient>> m_patient{};

public:
    Doctor(const std::string& name) :
        m_name{ name }
    {
    }

    void addPatient(Patient& patient);

    // We'll implement this function below Patient since we need Patient to be defined at that point
    friend std::ostream& operator<<(std::ostream& out, const Doctor& doctor);

    const std::string& getName() const { return m_name; }
};
```

Como se puede apreciar en este trozo de código (incompleto), como Doctor y paciente tienen una relación, en la declaración de Doctor ya se hace referencia al paciente con el método “AñadirPaciente()” debido a que este debe existir para “complementar a doctor”.

C#:

- Constructores

```cpp
public class Taxi
{
    public bool IsInitialized;

    public Taxi()
    {
        IsInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        Console.WriteLine(t.IsInitialized);
    }
}
```

Como podemos ver en el ejemplo, es muy sencillo identificar el constructor de la clase Taxi, siendo simplemente llamada y añadiendo los atributos para dicha declaración de constructor.

- Herencia

```
public class A
{
    private int _value = 10;

    public class B : A
    {
        public int GetValue()
        {
            return _value;
        }
    }
}

public class C : A
{
    //    public int GetValue()
    //    {
    //        return _value;
    //    }
}

public class AccessExample
{
    public static void Main(string[] args)
    {
        var b = new A.B();
        Console.WriteLine(b.GetValue());
    }
}
```

En C#, en vez de utilizar extends como otros lenguajes, se utiliza el símbolo “:” que funciona de la misma forma, pudiendo acceder a los métodos y o atributos de la clase referenciada tras dicho símbolo.

- Polimorfismo

```
public class Shape
{
    // A few example members
    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }

    // Virtual method
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Code to draw a circle...
        Console.WriteLine("Drawing a circle");
        base.Draw();
    }
}
public class Rectangle : Shape
{
    public override void Draw()
    {
        // Code to draw a rectangle...
        Console.WriteLine("Drawing a rectangle");
        base.Draw();
    }
}
public class Triangle : Shape
{
    public override void Draw()
    {
        // Code to draw a triangle...
        Console.WriteLine("Drawing a triangle");
        base.Draw();
    }
}
```

El polimorfismo en C# funciona de forma similar a otros lenguajes también, se crea una clase madre inicial que luego puede ser modificada de diferentes formas por clases hijas, tal y como se puede apreciar en el ejemplo.

- Sobrecarga de métodos

```
class SillyMath
{
    public static int Plus(int number1, int number2)
    {
        return Plus(number1, number2, 0);
    }

    public static int Plus(int number1, int number2, int number3)
    {
        return number1 + number2 + number3;
    }
}
```

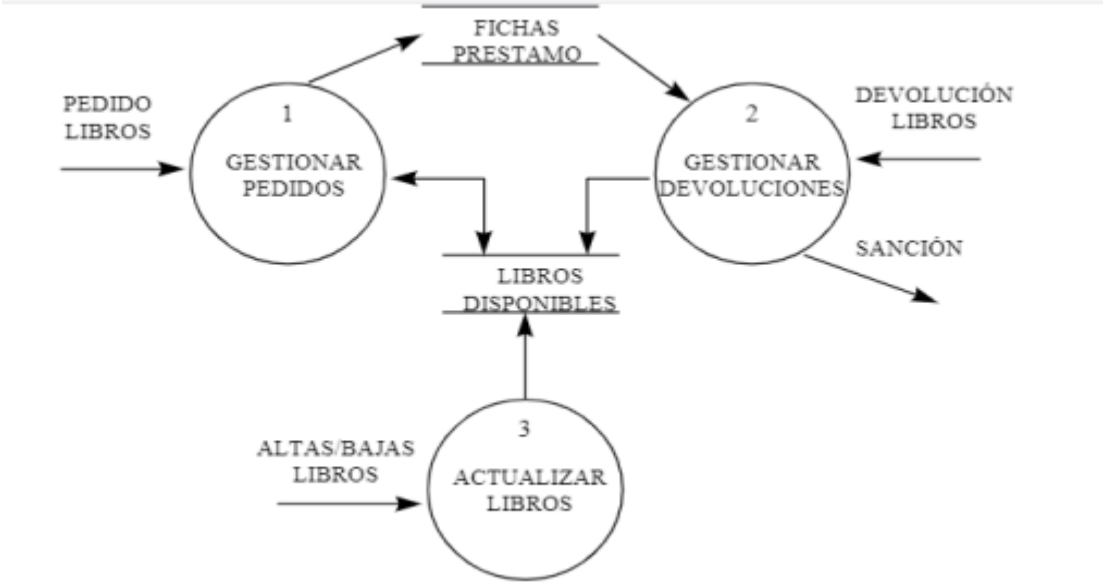
La sobrecarga de métodos se comporta de una forma muy similar en la gran mayoría de lenguajes, pudiendo modificar los métodos como el ejemplo mostrado.

- Asociaciones

```
public class Association
{
    //SomeUtilityClass objSC   /*NO local reference
    public void doSomething(SomeUtilityClass obj)
    {
       obj.DoSomething();
    }
}
```

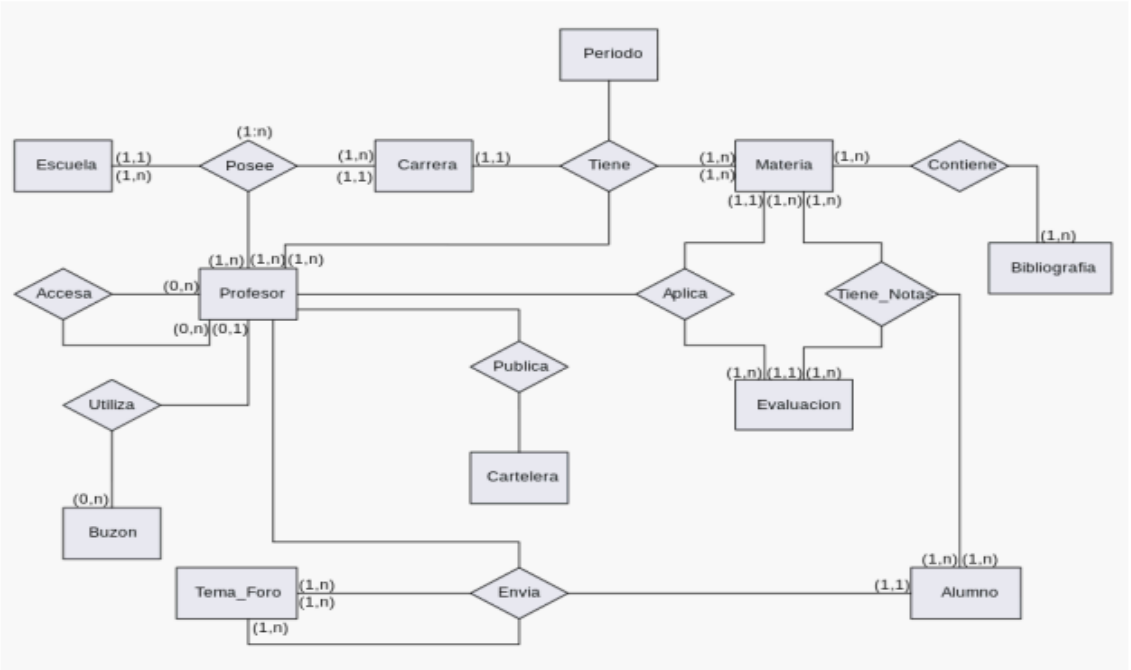
En el caso de las asociaciones es tan simple como muestra el ejemplo, crear un método y asignarle un objeto al que atribuirle dicho método.

Práctica 5.2: Escribe un enunciado que se corresponda con el DFD siguiente:



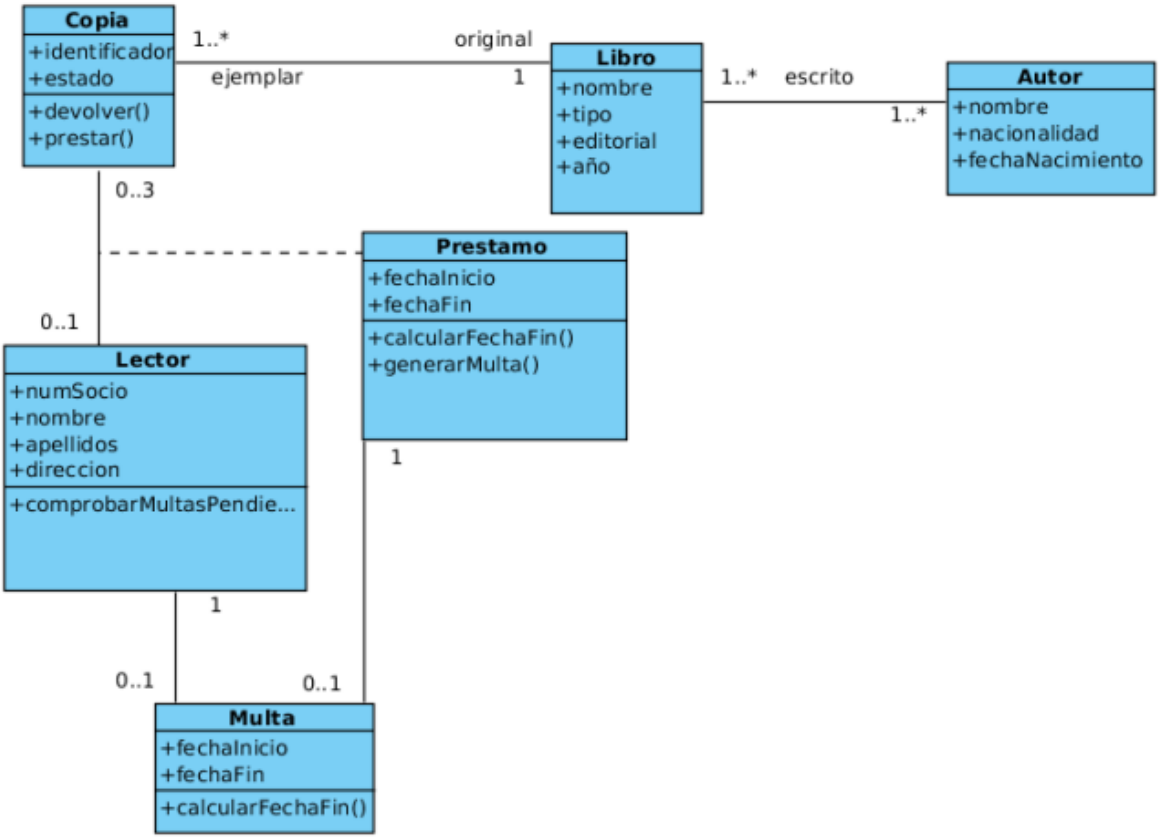
Cuando se realiza un pedido de libros, se tiene que gestionar los pedidos comprobando la disponibilidad de los libros y luego debe figurar en su ficha de préstamo correspondiente, donde también se agregan los datos de devolución.
Cuando se realiza una devolución, se debe comprobar la disponibilidad de los libros otra vez y recibir la información de la ficha de préstamo del libro a devolver para saber si se produce una sanción o no.
Cada vez que se pide o se devuelve un libro se debe actualizar los libros disponibles dando de alta y de baja los libros.

Práctica 5.3: Escribe un enunciado que se corresponda con el diagrama E-R siguiente:



Una escuela posee una o varias carreras que tienen un período, uno o varios profesores que posee la misma escuela y una o varias materias que pueden tener una o varias bibliografías que las referencie.
Las materias tienen las notas que figuran en la evaluación, siendo estas notas de un o varios alumnos.
La escuela posee uno o varios profesores que pueden acceder a la misma, y también pueden utilizar sus buzones.
Los profesores aplican la materia de una o varias evaluaciones, publican la cartelera y envían un o varios temas foro al igual que los alumnos.

Práctica 5.4: Interpretación de un diagrama de clases:

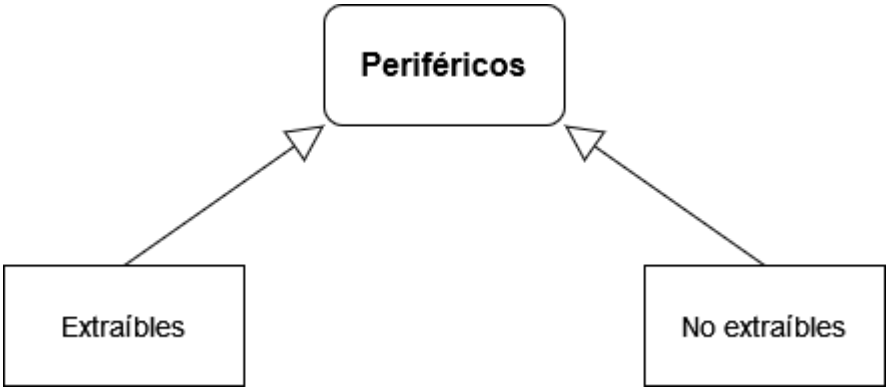


En una biblioteca figuran el o los autores junto a su nombre, nacionalidad y fecha de nacimiento de los diferentes libros con su respectivo nombre,tipo, editorial y año de publicación.
Cada libro puede tener una o varias copias que se diferencian por su estado y su identificador, y tienen sus respectivos métodos devolver() y prestar().
Solo se pueden prestar un máximo de 3 copias a 1 lector del que tenemos de información, su número de Socio, su nombre y apellidos y su dirección teniendo también el método comprobarMultasPendientes().
Cuando se presta una copia a un lector, se genera su respectivo préstamo, que se caracteriza por su fecha de inicio y de fin, con los método calcularFechaFin() y generarMulta().
La multa puede ser generada o no al lector junto a el préstamo, con sus respectivas fecha de inicio y fin y un método calcularFechaFin().

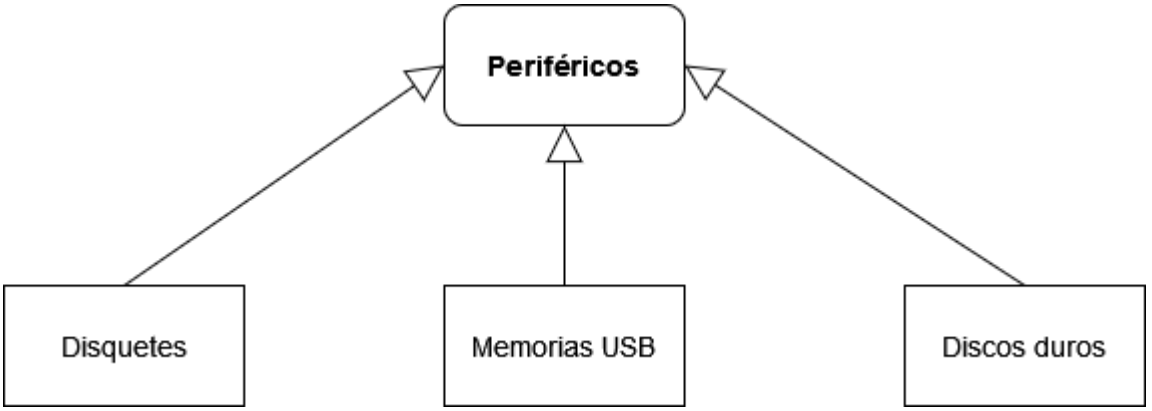
Práctica 5.5: Elaboración de diagramas de clases sencillos:

Representa mediantes distintos diagramas de clase independientes los siguientes escenarios:

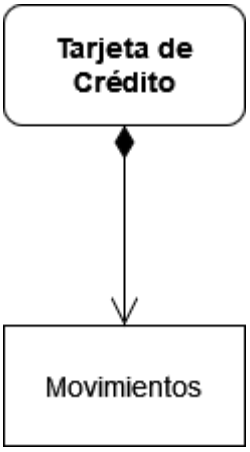
a) **Los periféricos que pueden ser extraíbles y no extraíbles.**



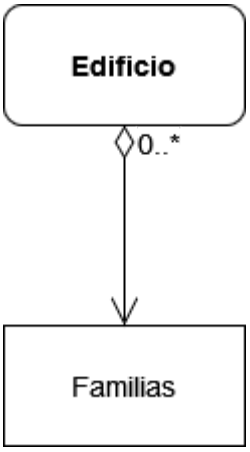
b) **Hay varios tipos de periféricos, por ejemplo: los disquetes, memorias USB y los discos duros.**



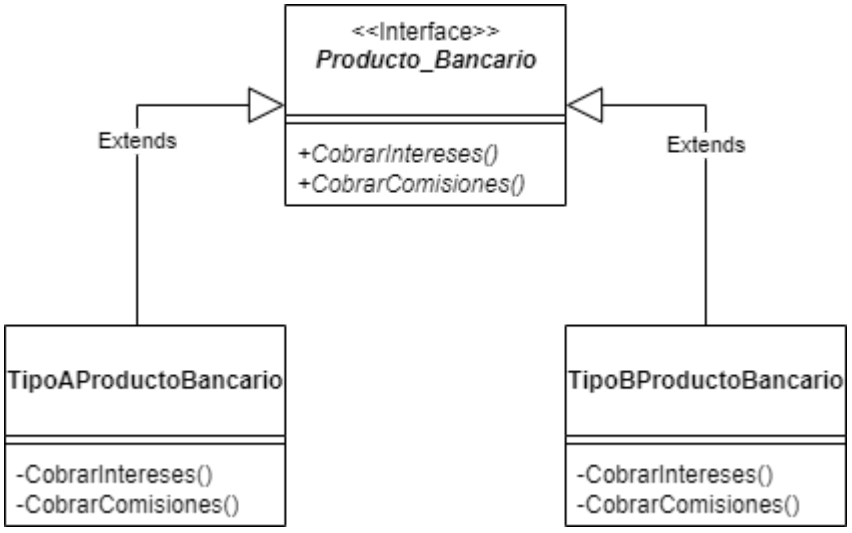
c) **Una tarjeta de crédito tiene una serie de movimientos.**



d) En un edificio viven varias familias (también puede estar vacío).

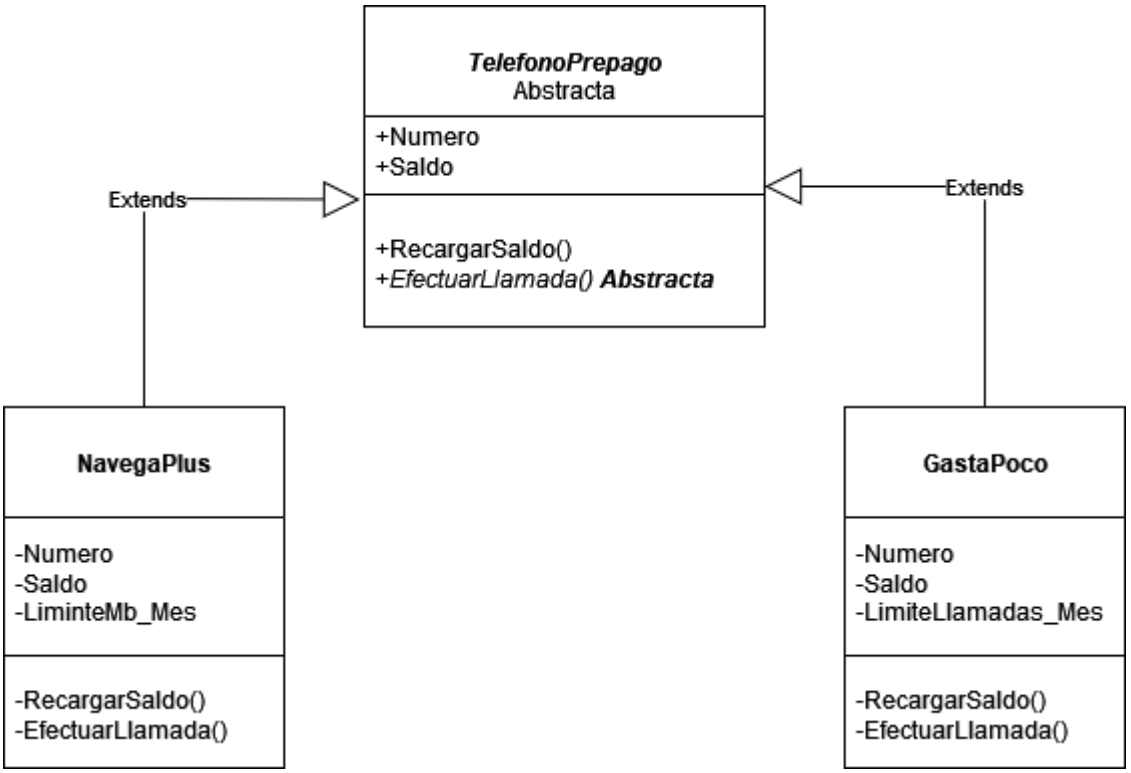


e) Hay varios tipos de productos bancarios, cada uno con sus atributos y métodos pero todos ellos deben de tener un método para cobrar intereses y cobrar comisiones aunque cada uno podrá utilizar una fórmula distinta.

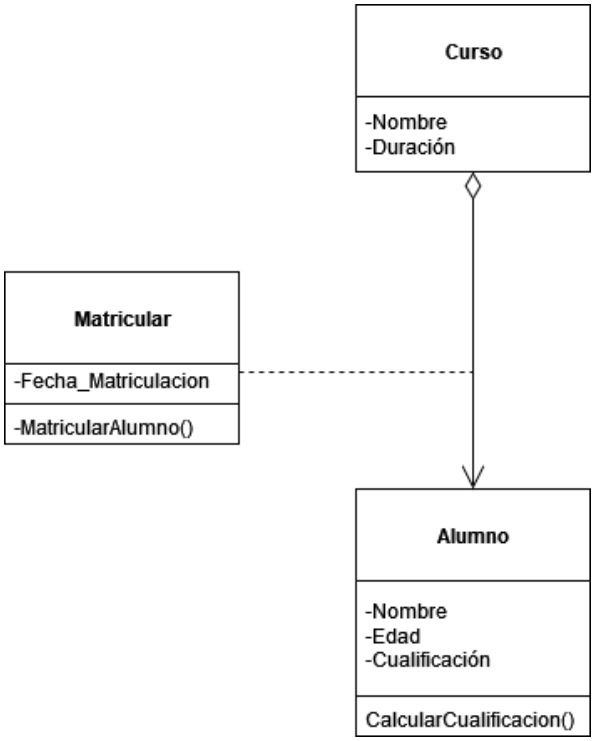


f) Los teléfonos de prepago tienen un número único, un saldo y una operación para recargar que es igual para todos los tipos de teléfono. Existen actualmente dos tipos de teléfono prepago: NavegaPlus (tiene un atributo con el límite de megas mensual) y GastaPoco (Con límite de llamadas mensuales). También deben de tener todos una operación para efectuar llamadas pero es distinta para cada tipo de teléfono ya que reduce el saldo con distintas fórmulas. No se pueden crear

instancias de teléfono, habrá que hacerlo de algún tipo en concreto: GastaPoco, NavegaPlus, etc.



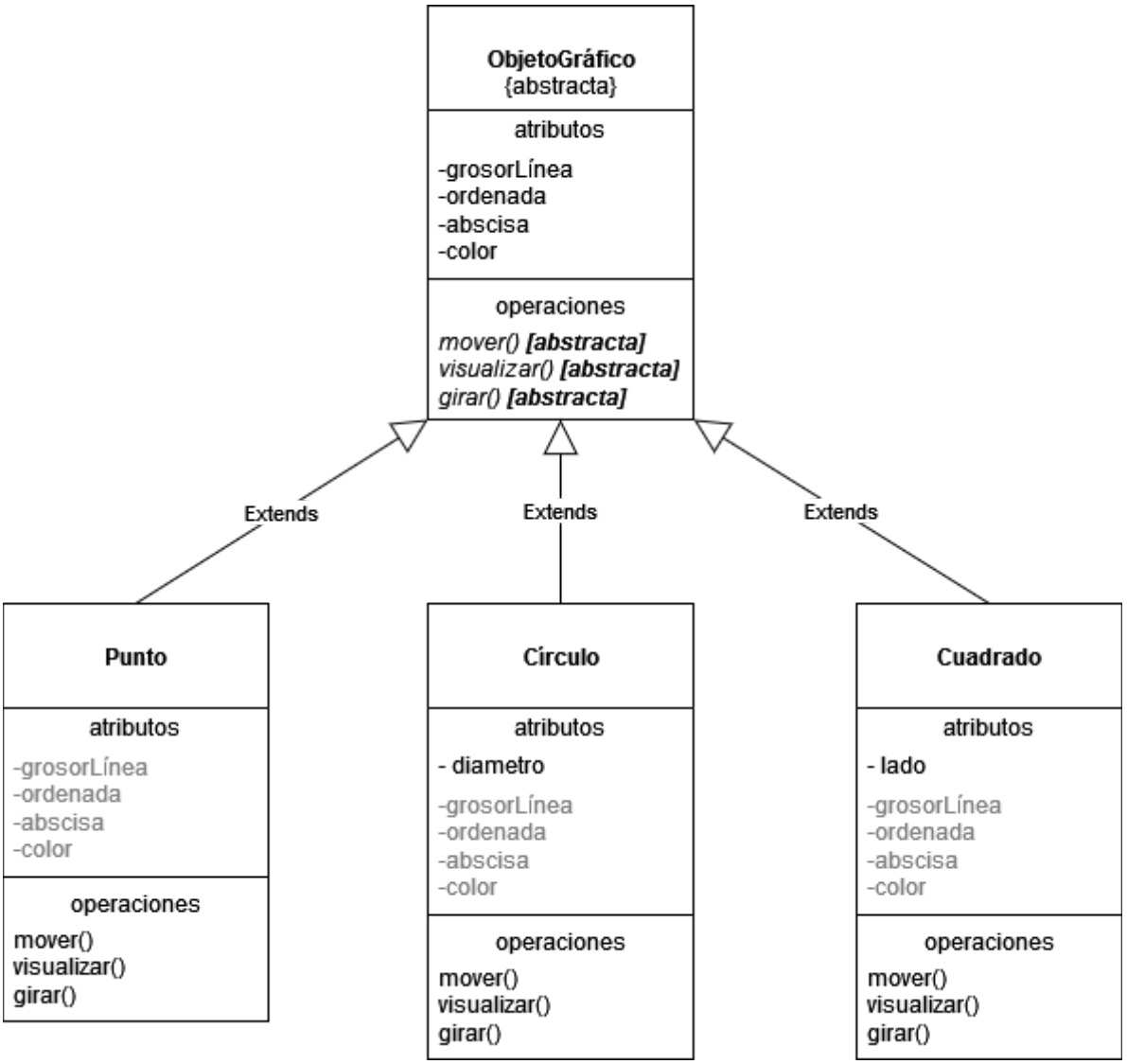
g) Los alumnos pueden matricularse en distintos cursos de una academia. De cada alumno queremos saber su nombre y edad. De cada curso su nombre y duración. También queremos saber la fecha en la que se matricula un alumno en un curso y su cualificación obtenida, habiendo una operación para calcular dicha cualificación.



Práctica 5.6: Realización de un diagrama de clases:

Representa mediante diagramas de clase lo siguiente:

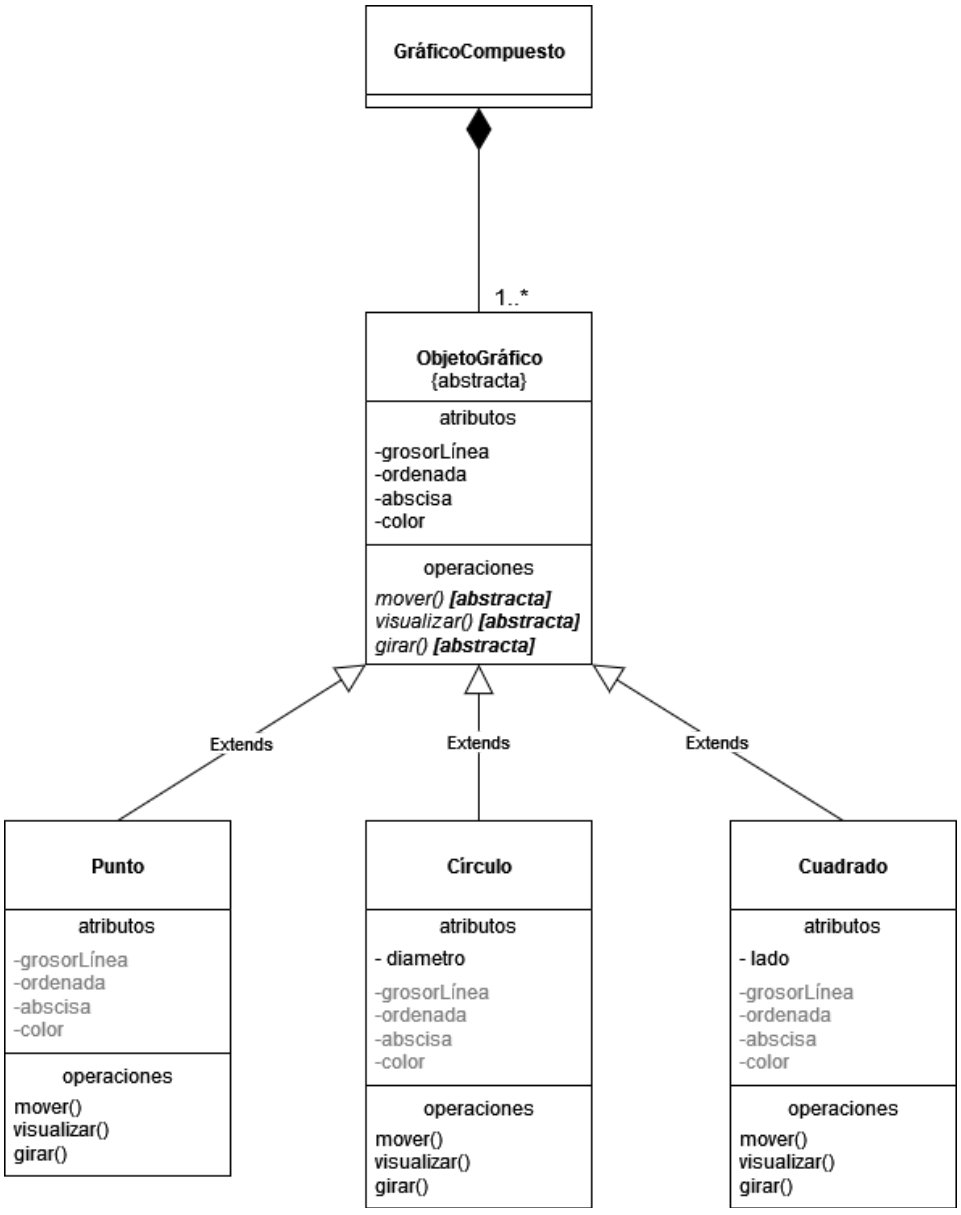
- **Una superclase ObjetoGráfico que tiene como atributos protegidos: grosorLínea, ordenada,abscisa, color y como operaciones públicas: mover(), visualizar() y girar().**
- **Tres subclases Punto, Círculo (tiene atributo privado diámetro) y Cuadrado (tiene atributo privado lado). Cada una de estas clases tiene una manera diferente de moverse, visualizarse y girar.**



- **Contesta las siguientes preguntas:**
  - **¿Es accesible el diámetro desde Cuadrado?**
    No, porque el diámetro es un atributo privado exclusivo de Círculo.

- **¿Un objeto Círculo posee un atributo color?**
  Sí, lo hereda de la superclase ObjetoGráfico.
- **¿Puede aplicarse el método mover a un objeto Punto?**
  Sí, pero debe establecerse debido a que lo hereda de forma abstracta.
- **¿Qué interés puede tener que la clase ObjetoGráfico sea abstracta?**
  Que las operaciones de cada subclase sean diferentes pero que dichas operaciones sean mandatorias.

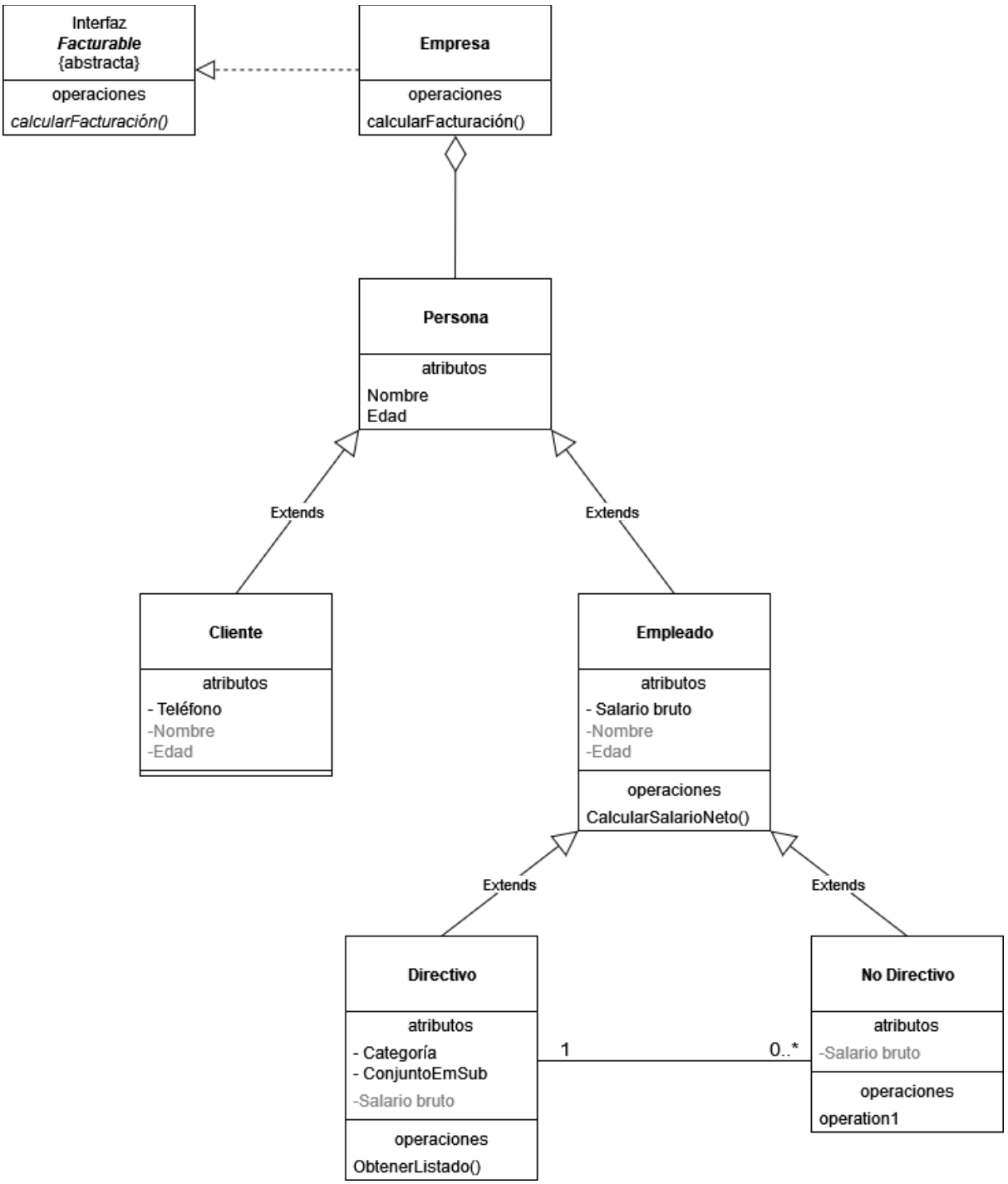
Extiende el modelo anterior para que un nuevo objeto gráfico llamado GráficoCompuesto esté compuesto de varios objetos gráficos, de tal forma que la supresión del objeto GráficoCompuesto implique la supresión de los ObjetoGráfico que lo componen.



Práctica 5.7: Elaboración de un diagrama de clases:

Representa a través de un diagrama de clases el siguiente escenario:

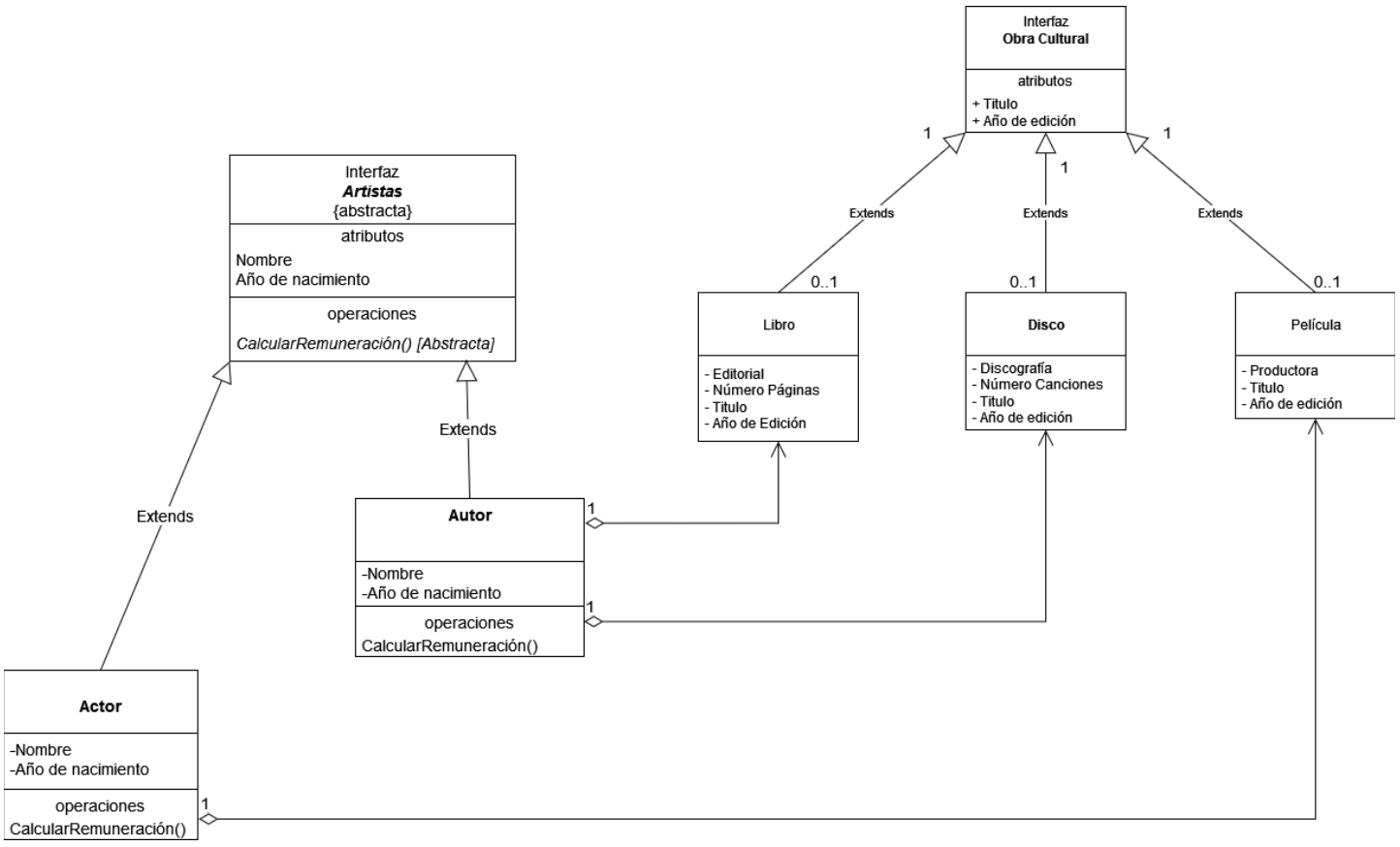
- **Necesitamos desarrollar una aplicación para gestionar la información de diferentes empresas, de sus clientes y de sus empleados.**
- **Tanto de clientes como de empleados almacenaremos su nombre y su edad.**
- **De los empleados almacenaremos su salario bruto y la aplicación tendrá que calcular el salario neto. Los empleados que son directivos tienen una categoría así como un conjunto de empleados subordinados.**
- **De los clientes además se necesita conocer su teléfono de contacto.**
- **Para un directivo, se puede obtener el listado de sus empleados subordinados.**
- **De las empresas no tenemos ningún atributo ya que habrá varios tipos (por ejemplo autónomo, PEME, holding, etc.) y los atributos y operaciones serán diferentes para cada uno de ellos. Pero cada uno de esos tipos de empresa deberá desarrollar un método llamado *calcularFacturación,* que dependerá de sus características.**


Interfaz
Facturable
{abstracta}
operaciones
calcularFacturación()
Empresa
operaciones
calcularFacturación()
Persona
atributos
Nombre
Edad
Extends
Extends
Cliente
atributos
- Teléfono
-Nombre
-Edad
Empleado
atributos
- Salario bruto
-Nombre
-Edad
operaciones
CalcularSalarioNeto()
Extends
Extends
Directivo
atributos
- Categoría
- ConjuntoEmSub
-Salario bruto
operaciones
ObtenerListado()
1
0..*
No Directivo
atributos
-Salario bruto
operaciones
operation1

Práctica 5.8: Elaboración de un diagrama de clases:

Representa a través de un diagrama de clases el siguiente escenario, especificando los tipos de datos que consideres apropiados para los atributos:

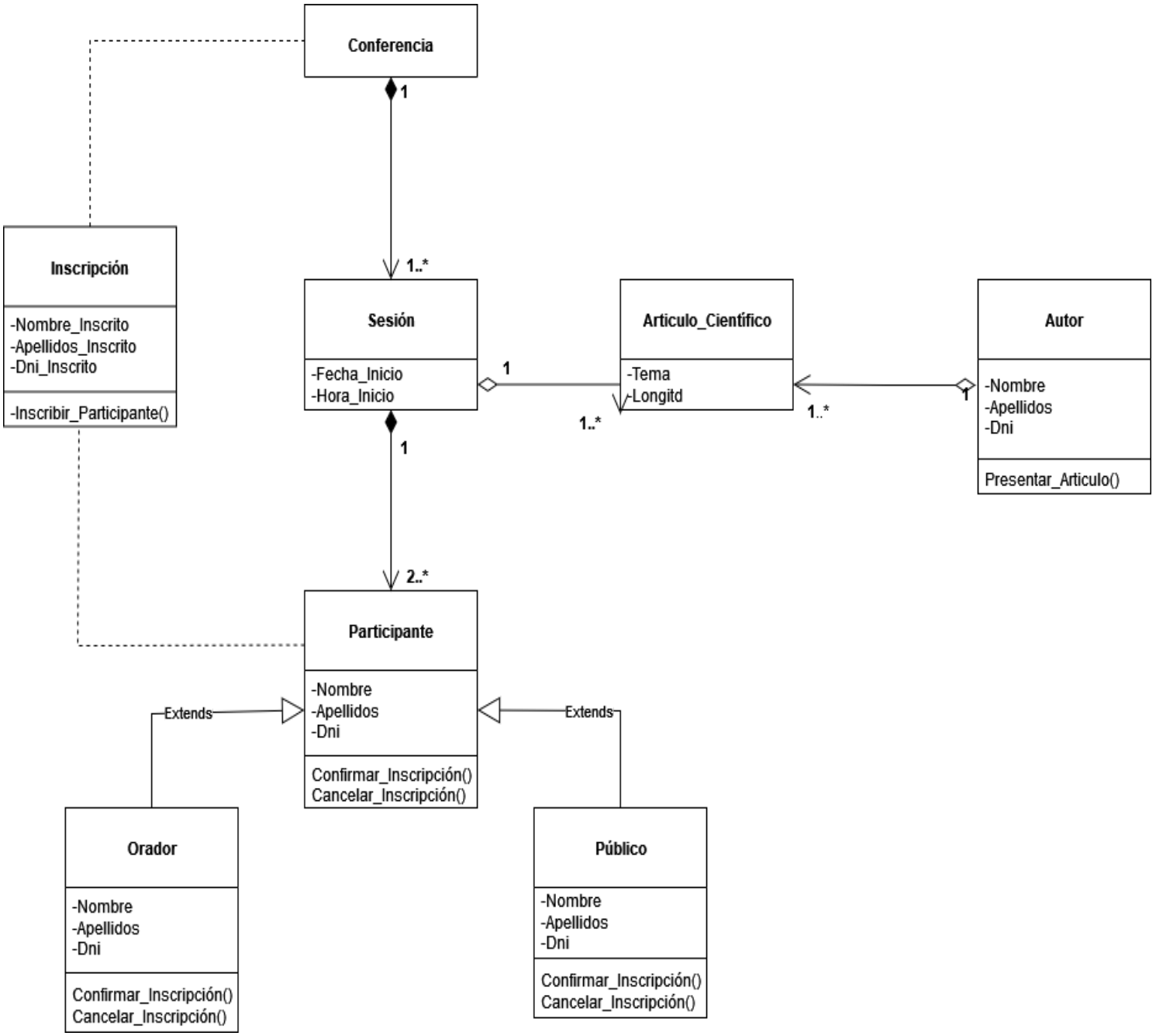
- **Se desea almacenar la información de distintos tipos de obras culturales. Dichas obras pueden ser libros, discos o películas.**
- **De toda obra nos interesa almacenar su título y año de edición.**
- **De los libros nos interesa almacenar su editorial y su número de páginas**
- **De los discos nos interesa almacenar su discográfica y el número de canciones.**
- **De las películas nos interesa almacenar su productora.**
- **Además nos interesa almacenar los distintos artistas relacionados con las obras, en concreto los autores que crearon la obra y, en caso de las películas, los actores que la interpretan.**
- **De los artistas nos interesa su nombre y año de nacimiento y todos deben tener una operación para calcular su remuneración pero depende de lo que sean: autores,actores, etc.**



Práctica 5.9: Elaboración de un diagrama de clases:

Representa mediante un diagrama de clase la gestión de una conferencia científica con las consideraciones siguientes:

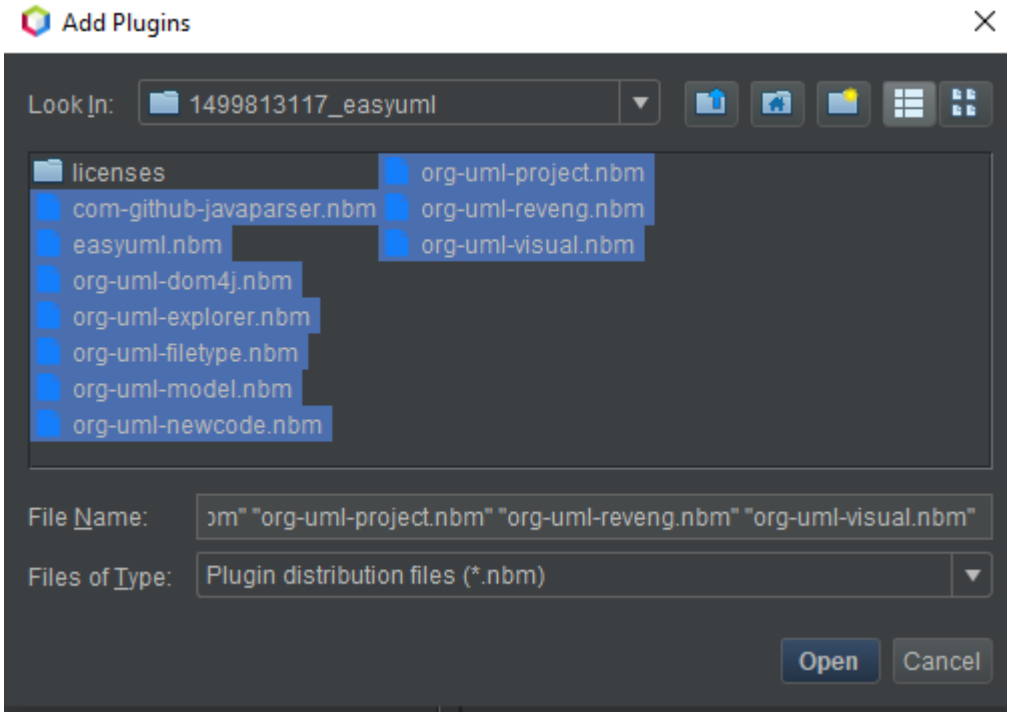
- **La conferencia puede tener varias sesiones.**
- **Una sesión tiene fecha y hora de inicio, pertenece solo a una conferencia y no tiene sentido sin una conferencia.**
- **Los participantes en una sesión pueden ser oradores o público. Todos ellos tienen que inscribirse en la conferencia. Puede cancelarse o confirmarse una inscripción También queremos guardar los datos con los que se inscribió en la sesión.**
- **Uno o más artículos científicos se presentan durante una sesión. Cada artículo puede ser corto o largo y trata de un tema determinado.**
- **Un autor puede tener uno o varios artículos presentados en la conferencia.**
- **Coloca los atributos que te parezcan convenientes**


Conferencia
1
1..*
Inscripción
-Nombre_Inscrito
-Apellidos_Inscrito
-Dni_Inscrito
-Inscribir_Participante()
Sesión
-Fecha_Inicio
-Hora_Inicio
1
1..*
Articulo_Científico
-Tema
-Longitd
1..*
1
Autor
-Nombre
-Apellidos
-Dni
Presentar_Articulo()
1
2..*
Participante
-Nombre
-Apellidos
-Dni
Confirmar_Inscripción()
Cancelar_Inscripción()
Extends
Extends
Orador
-Nombre
-Apellidos
-Dni
Confirmar_Inscripción()
Cancelar_Inscripción()
Público
-Nombre
-Apellidos
-Dni
Confirmar_Inscripción()
Cancelar_Inscripción()

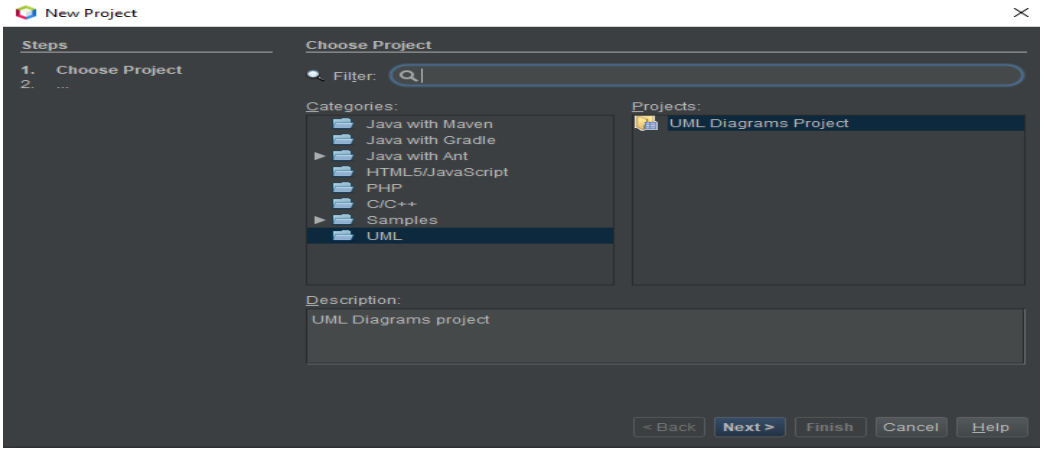
Práctica 5.10: Instalación y manejo del módulo UML de NetBeans:

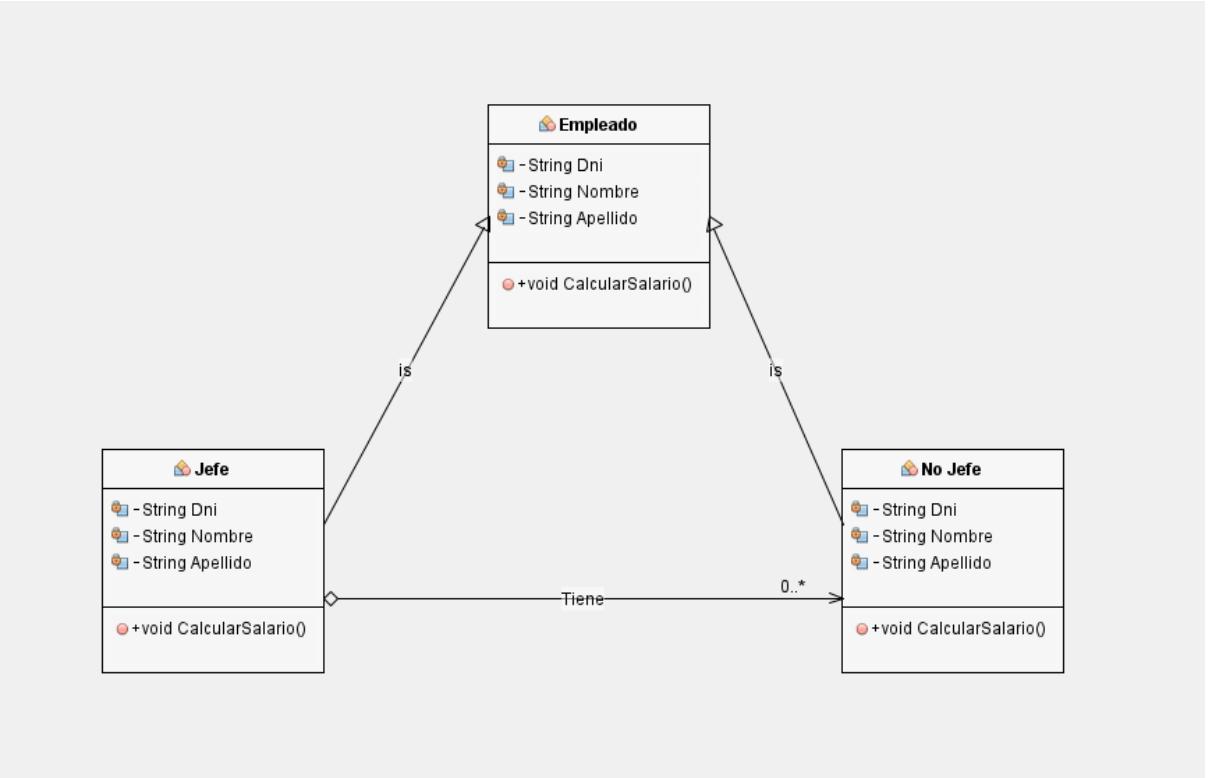
Utilizando tu contorno de desarrollo NetBeans, realiza los siguientes apartados:

- **Siguiendo las indicaciones dadas en el texto de la actividad, instala el módulo de UML en el contorno de desarrollo.**
  - Descargamos el .zip de Easyuml y lo extraemos.
  - Ahora vamos a Tools>Plugins y en la pestaña Downloaded le damos a Add Plugins seleccionando todo en nuestra carpeta de easyuml menos la carpeta de licencias:

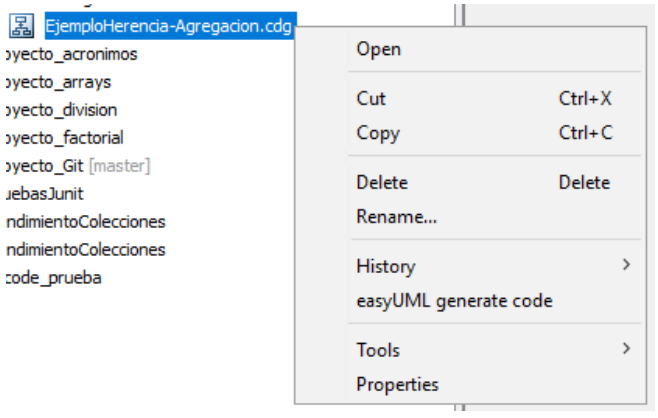


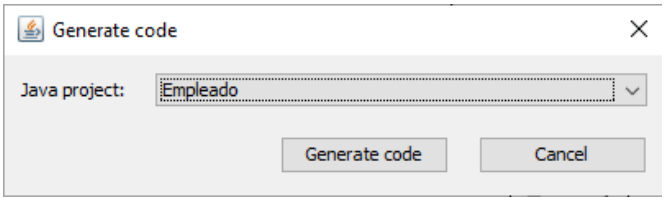
  - Luego le daremos a Open y a Install aceptando los términos y condiciones y confirmando la instalación.
- **Usando el módulo UML instalado inventa un diagrama de clases que incluya herencia y agregación.**
  - Ahora ya podemos crear un proyecto Uml desde New Project:



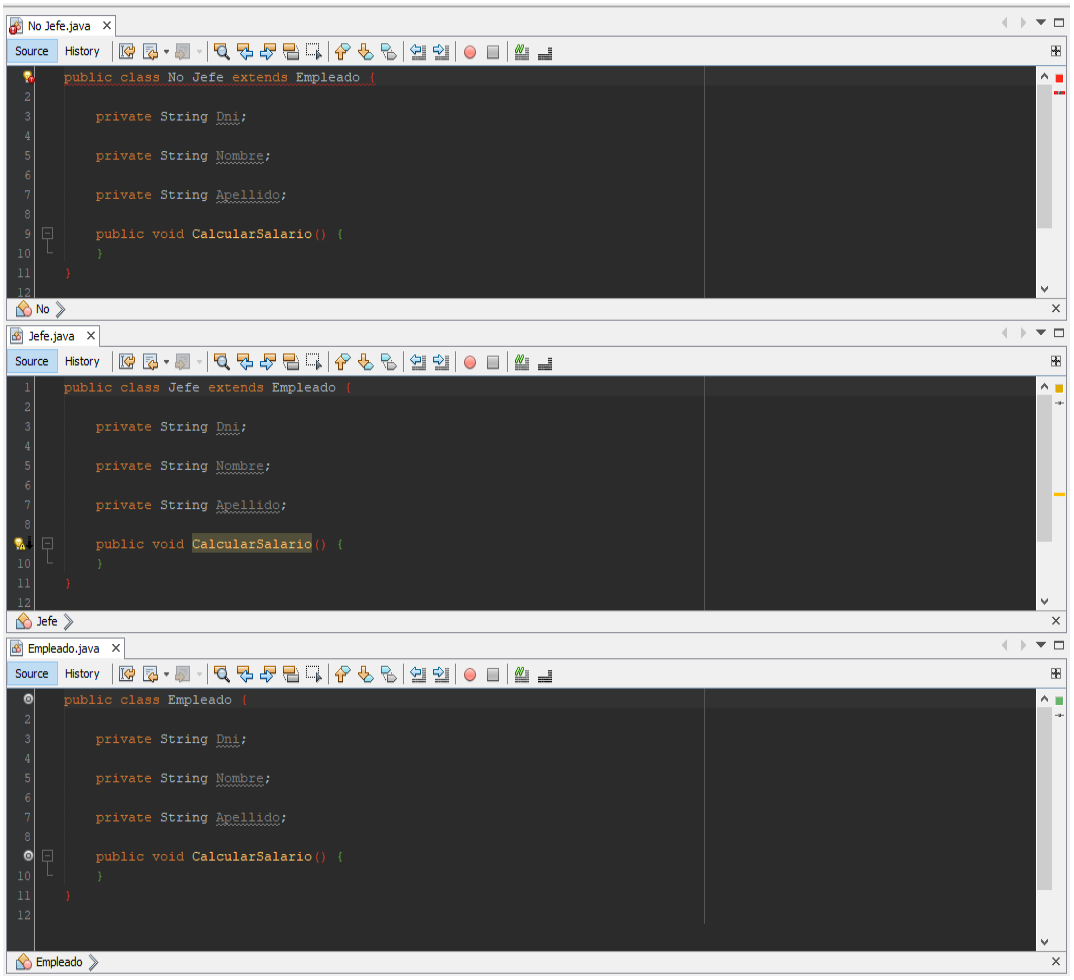


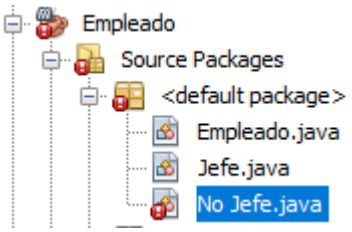
- **Crea en Netbeans un nuevo proyecto Java vacío y genera automáticamente código a partir del diagrama hecho previamente.**
  Creamos un nuevo proyecto java en File > Create Project en este caso de nombre Empleado.
  Hacemos click derecho en nuestro anterior diagrama y seleccionamos la opción easyUML generate code y seleccionamos nuestro proyecto creado:





Luego podremos comprobar en nuestro proyecto Empleado que se ha generado el código acorde a nuestro diagrama creado previamente:

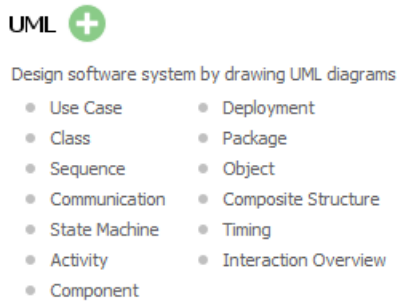




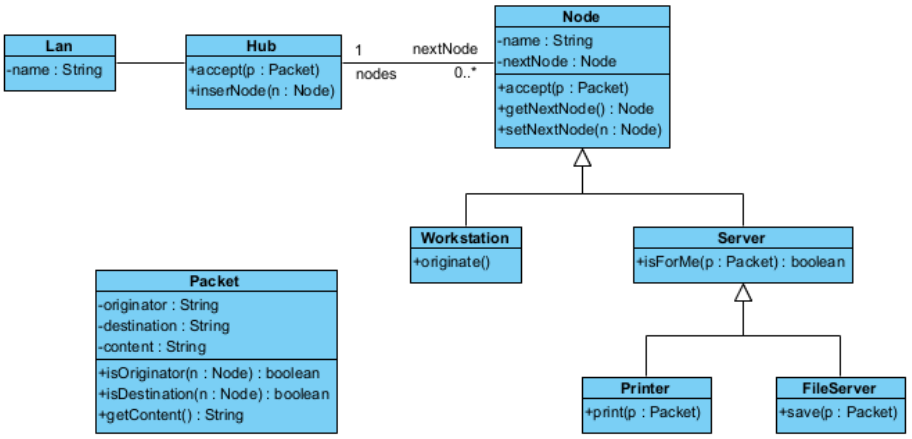
Práctica 5.11: Instalación y manejo de Visual Paradigm:

Siguiendo las indicaciones dadas en el texto de la actividad, utiliza la versión gratuita de Visual Paradigm y después realiza los diagramas de clases de las prácticas 5 a 9 de este tema.

- Instalamos Visual Paradigm Community Edition y proporcionamos o no un email.
- Una vez instalado podremos acceder a los diagramas UML:

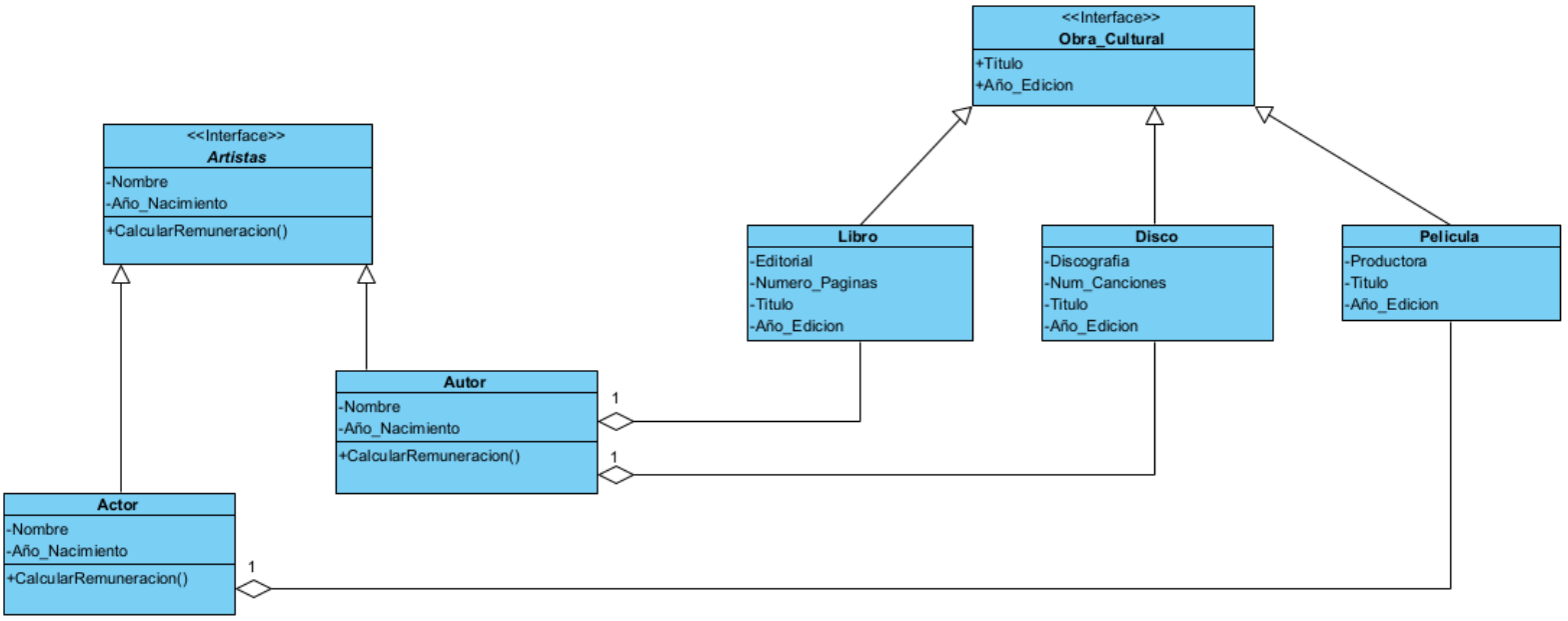


- Tendremos acceso a plantillas como la siguiente:



Ejemplos con Ejercicios 5.8 y 5.9:

Ejercicio 5.8:



Ejercicio 5.9: