ALGORITMOS Y PROGRAMACIÓN

Clase 6

Programación orientada a objetos: UML

Universidad Nacional ARTURO JAURETCHE

UML

- Un diseño Orientado a Objetos expresa, sin necesidad de escribir código, cómo colaboran los objetos para realizar sus actividades.

- Han existido diversos métodos y técnicas orientadas a objetos, con muchos aspectos en común pero utilizando distintas notaciones.

- Como resultado de varias de ellas, surgió el Lenguaje Unificado de Modelado (Unified Modeling Language – UML) como estándar para el modelado de sistemas de software. En 1994, Booch, Rumbaugh (OMT) y Jacobson (Objectory) deciden unificar sus métodos y nace el UML.

UML

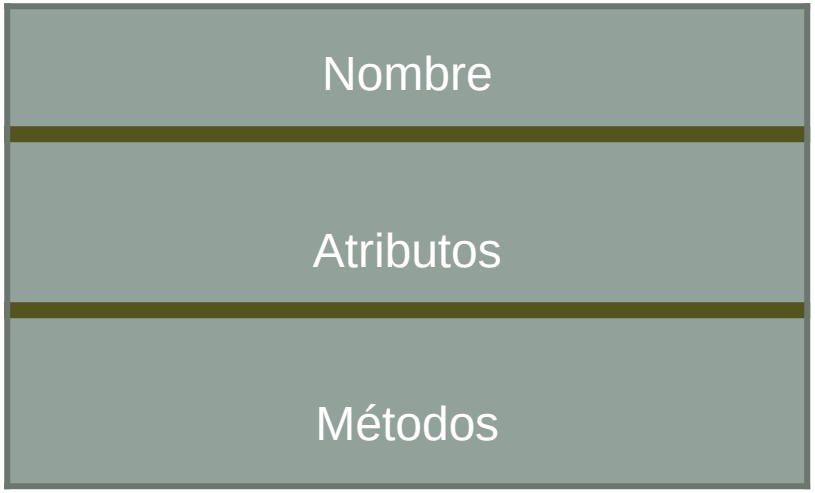
- Posee distintos tipos de diagramas, pero en este curso veremos sólo el Diagrama de Clases.

- Los diagramas de clase describen los tipos de objetos de un sistema, así como los distintos tipos de relaciones que pueden existir entre ellos.

- Dentro del diagrama de clases, una clase se describe en función de sus atributos y de sus métodos.

UML

- Gráficamente una clase se representa con un rectángulo con 3 sectores:
- Nombre de la clase
- Atributos
- Métodos



UML Ejemplo clase Vehículo

```
public class  Vehiculo

{    private int peso;

     protected int cantidadDePasajeros;

      public Vehiculo() {   }

     public void acelerar (double vel) { ….   }

    public void frenar () { …}

}
```

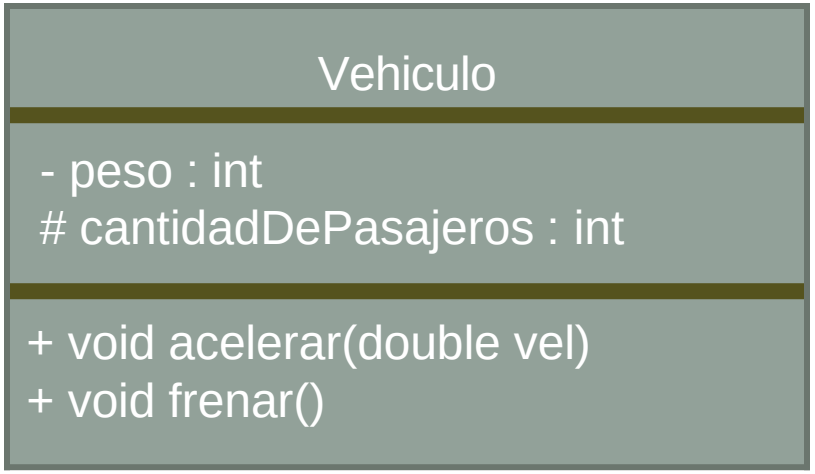


Diagrama de clases (UML)

Ejemplo clase Vehículo

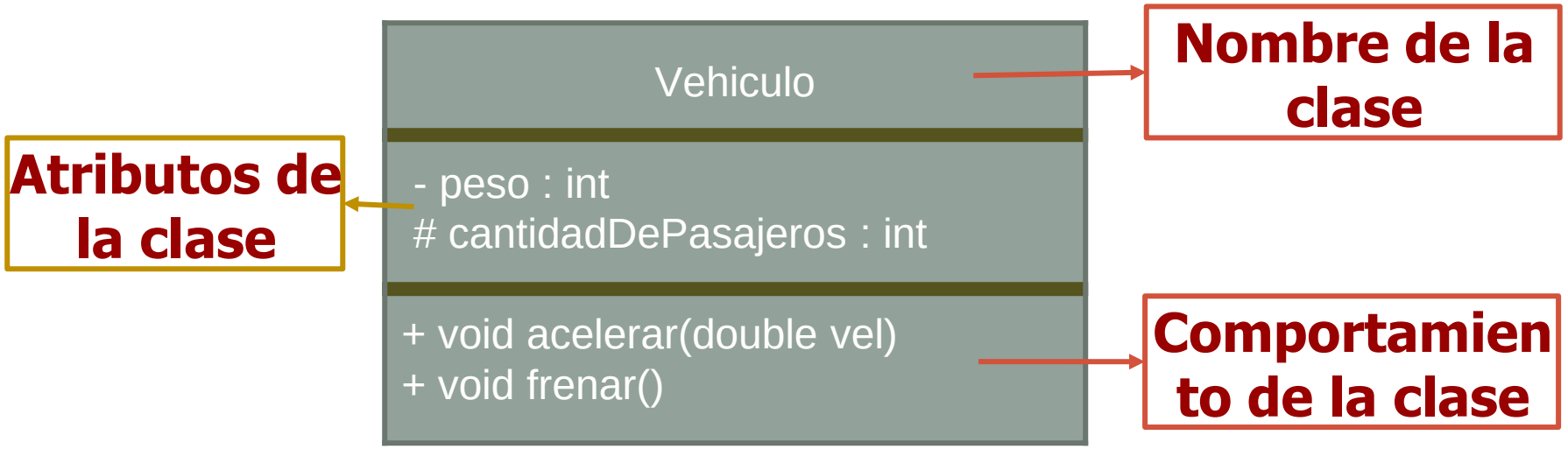
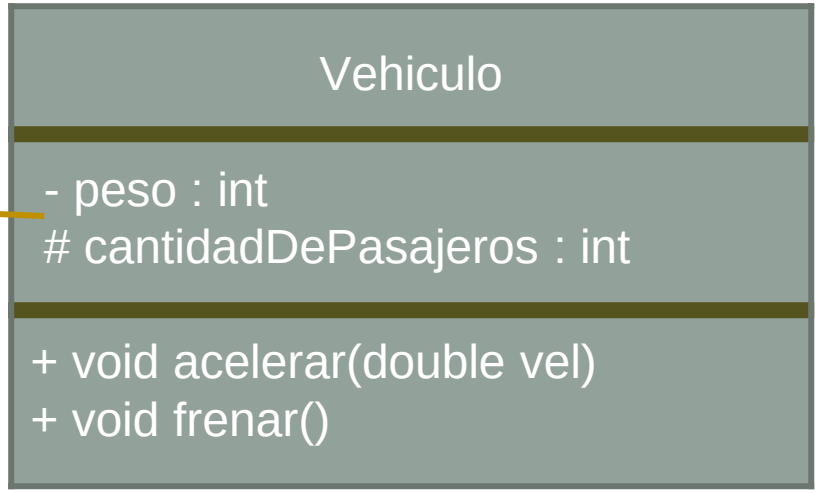


Diagrama de clases (UML)

Ejemplo clase Vehículo



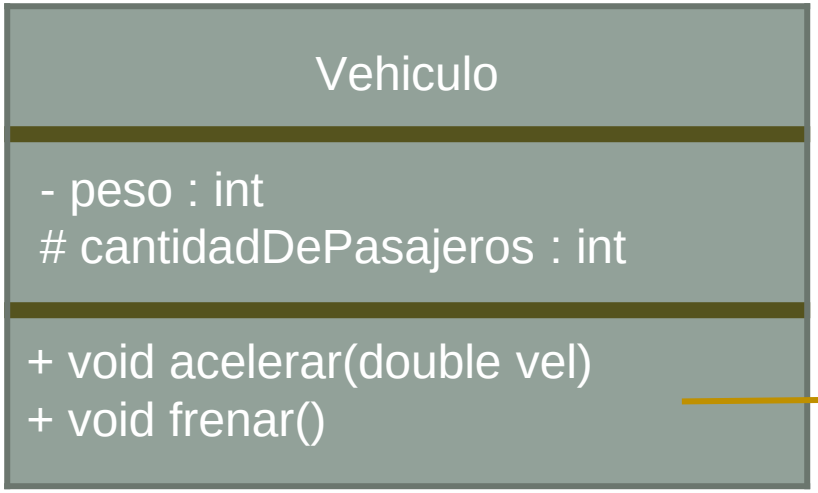
Atributos de la clase

Se establece el nombre del atributo, su tipo y su nivel de protección:

- Atributo privado
+ Atributo público
 # Atributo protegido

Diagrama de clases (UML)

Ejemplo clase Vehículo

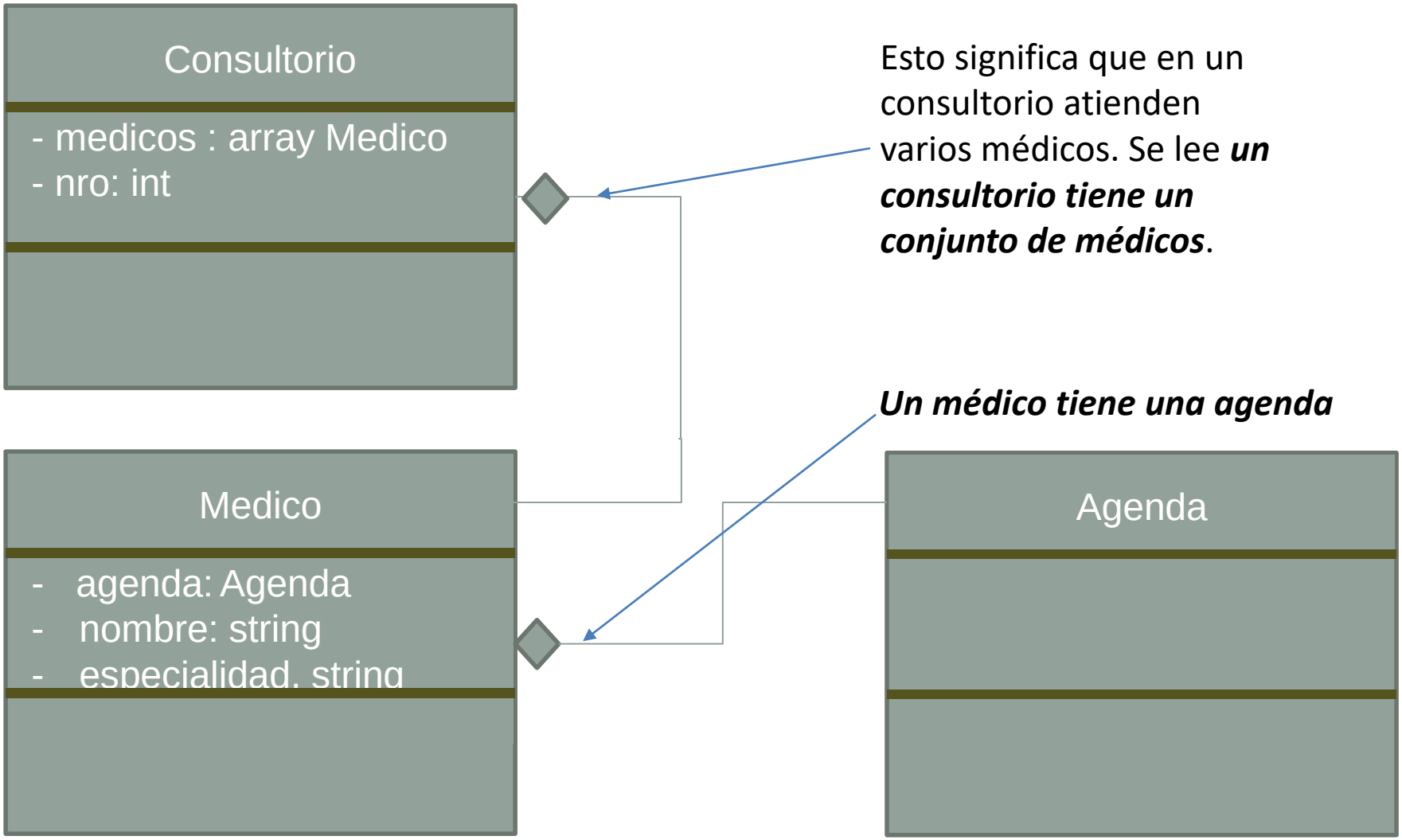


Comportamiento de la clase

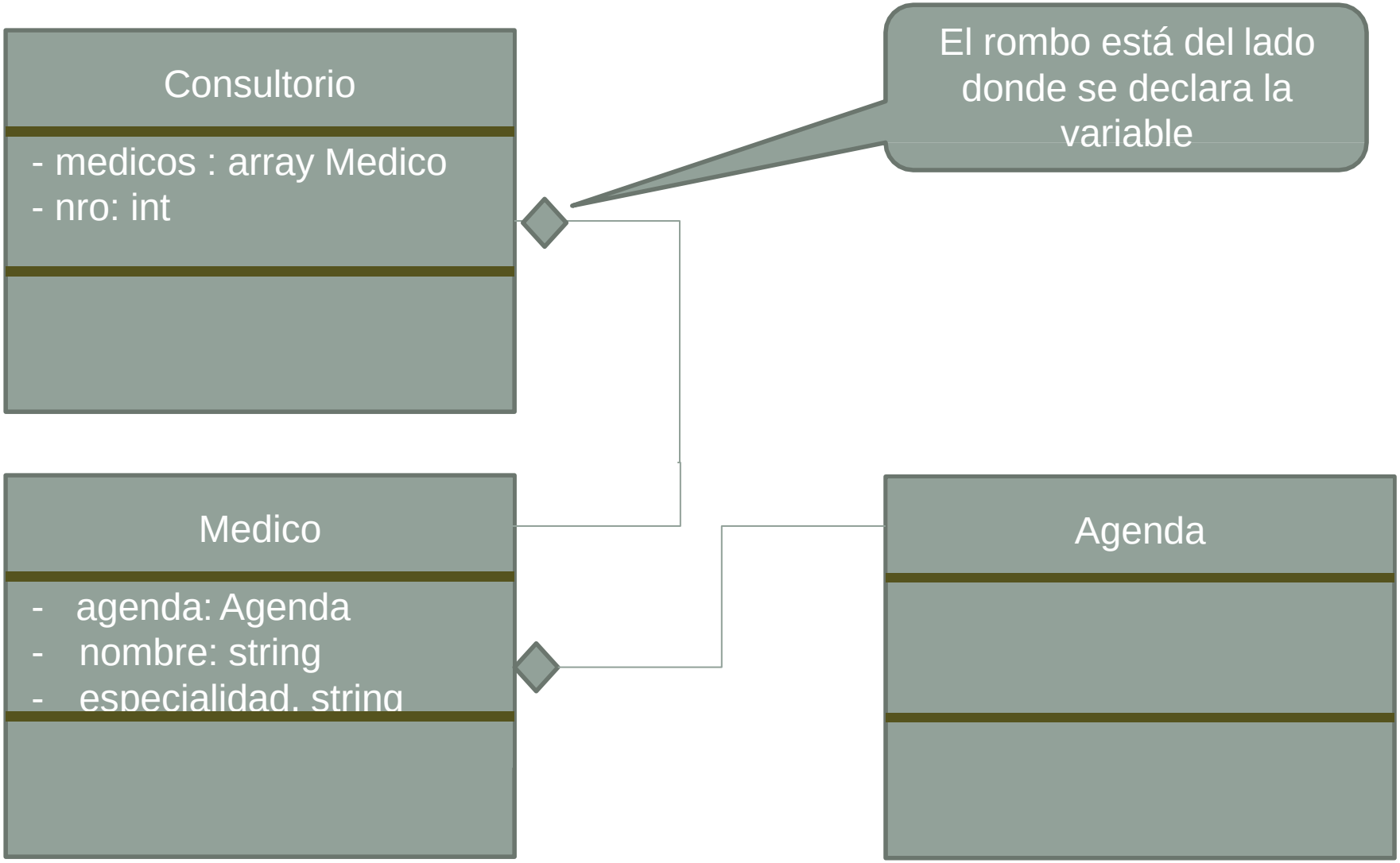
Se establece el nombre del método, la lista de parámetros que recibe, el tipo del valor de retorno y su nivel de protección:

- Método privado
+ Método público
 # Método protegido

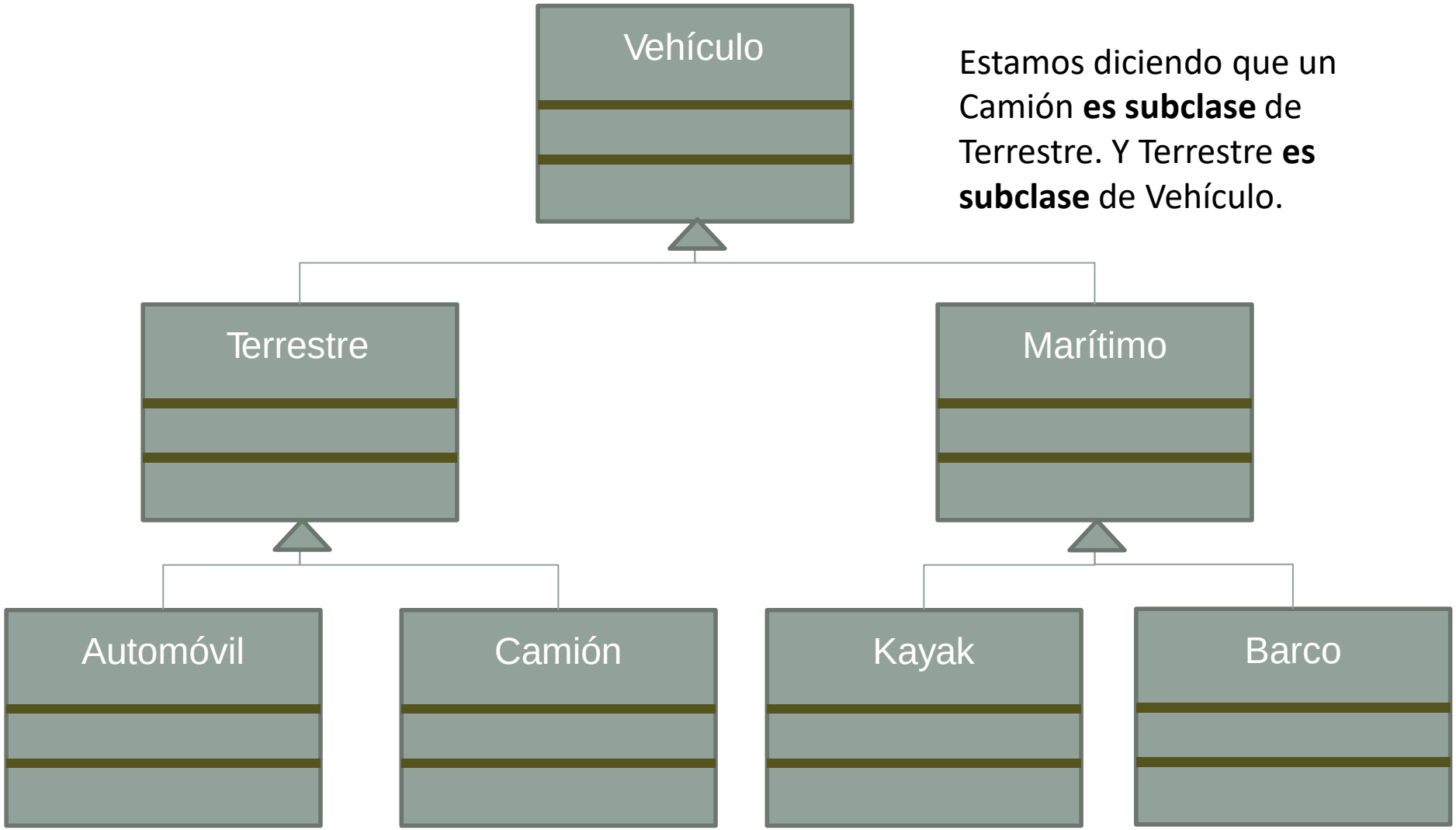
Relaciones entre los objetos: Composición



Relaciones entre los objetos: Composición

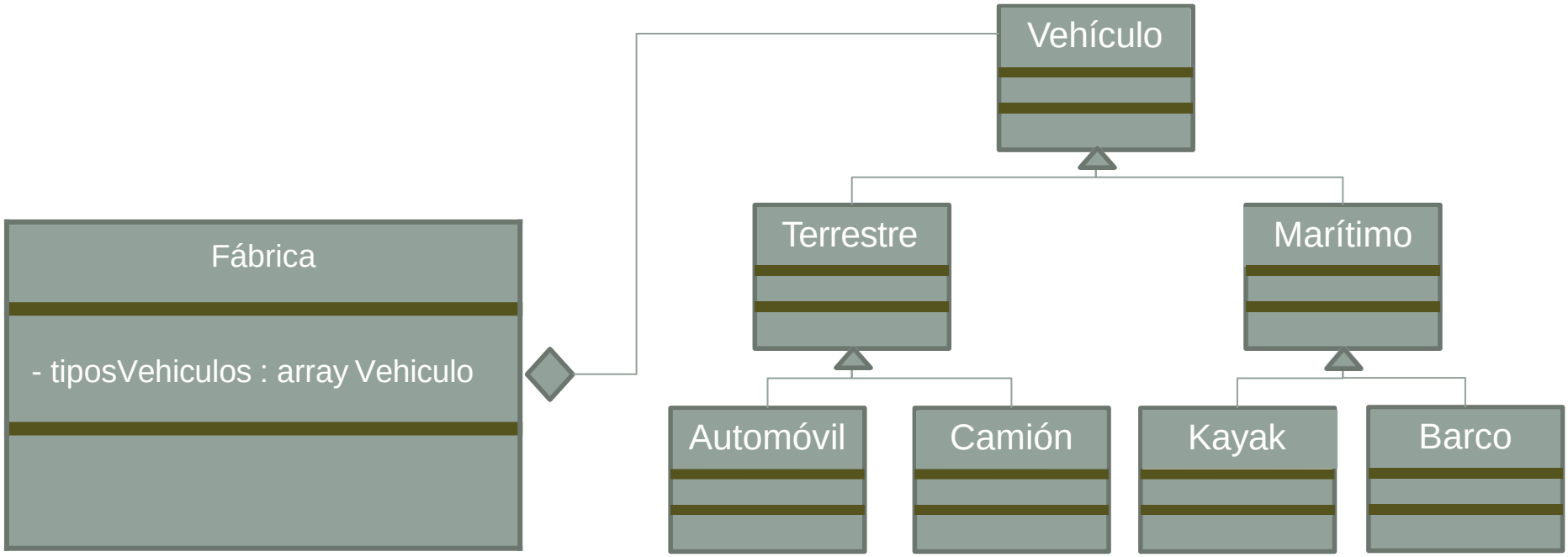


Relaciones entre los objetos: Herencia o clasificación



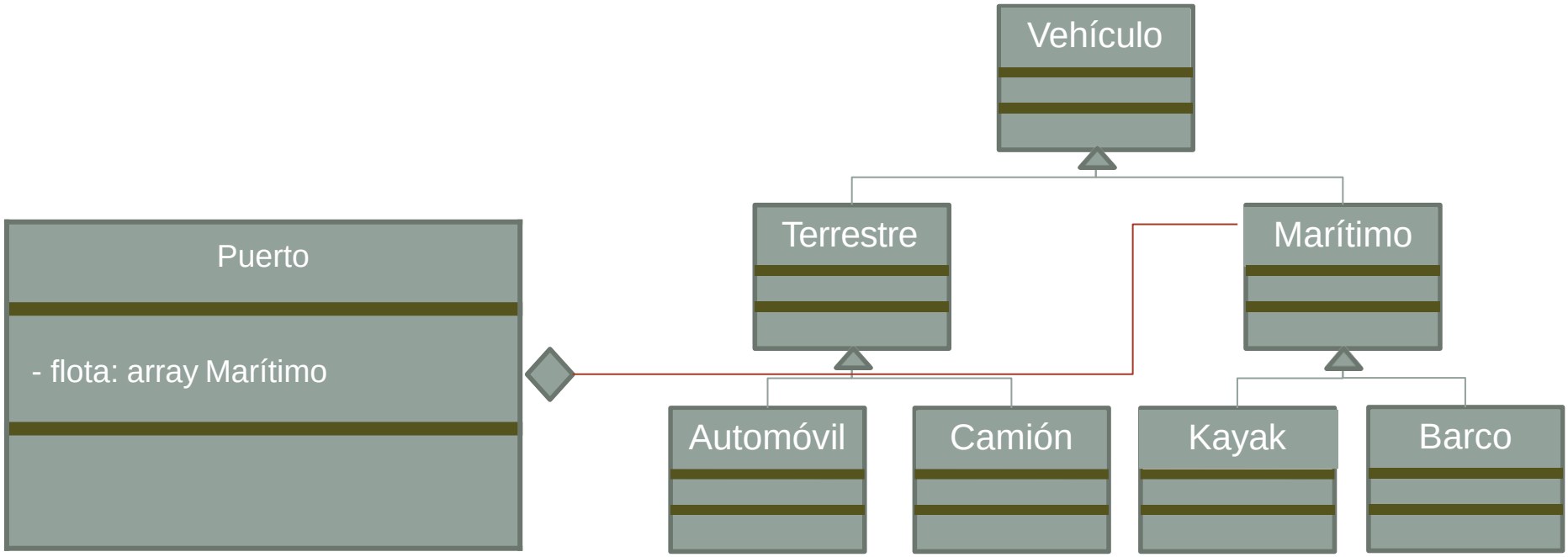
Estamos diciendo que un Camión **es subclase** de Terrestre. Y Terrestre **es subclase** de Vehículo.

Herencia y composición



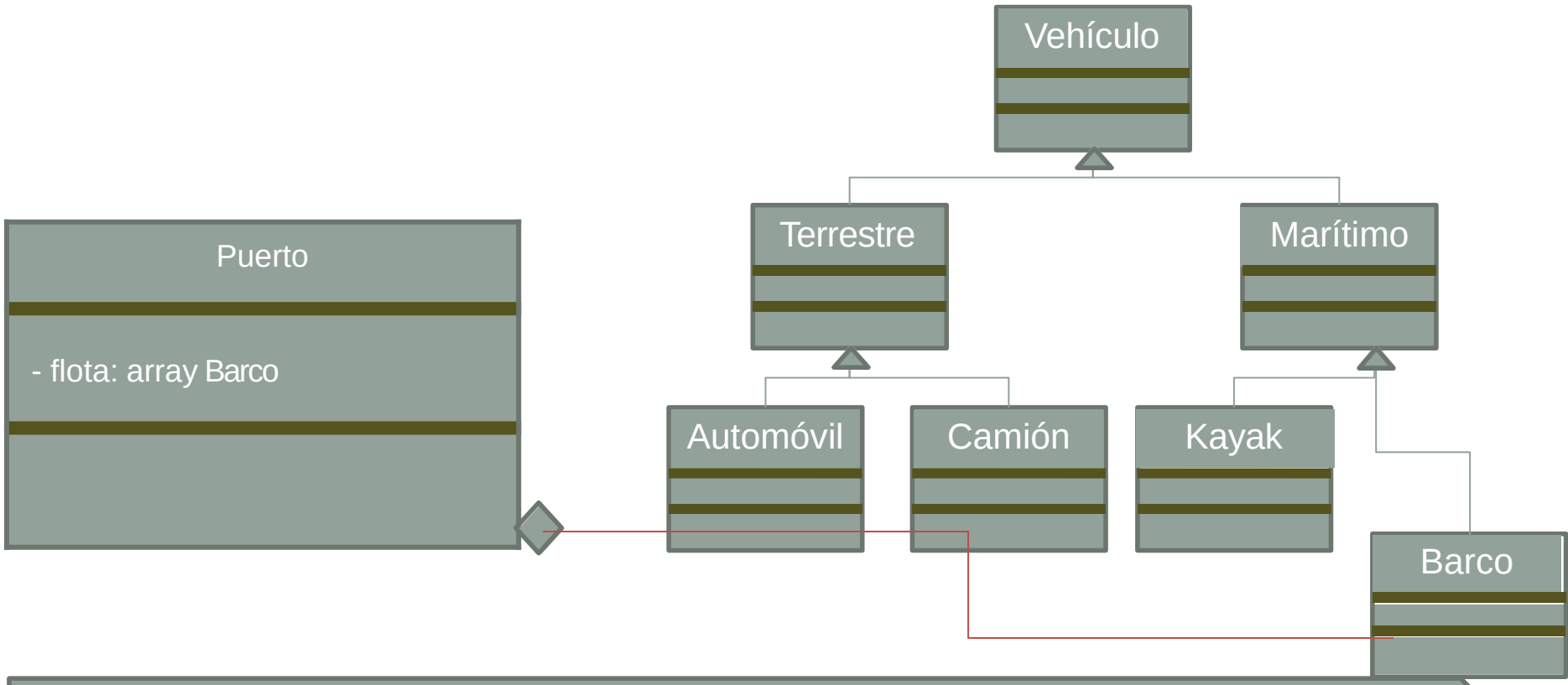
Estamos diciendo que en la variable tiposVehiculos de la clase Fábrica puede haber cualquier instancia de Vehículo o de sus subclases.
Una Fabrica tiene un conjunto de vehiculos (de distintas clases)

Herencia y composición



Estamos diciendo que en la variable flota de la clase Puerto puede haber cualquier instancia de Marítimo o de sus subclases

Herencia y composición

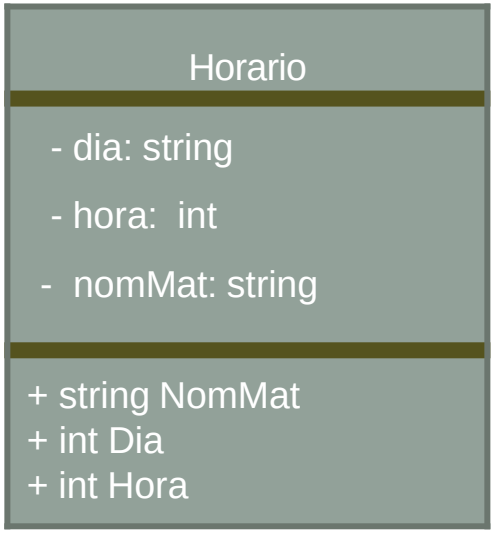


Estamos diciendo que en la variable flota de la clase Puerto puede haber solo instancias de la clase Barco.

Puesta en común:

1) Implemente la clase Alumno para modelar alumnos en una Universidad. Agréguele como estado lo que considere necesario para el problema. Piense ¿cómo implementaría los horarios de cursada de un alumno? Considere que debe guardar el horario (día, hora) y el nombre de la materia a cursar en ese horario.

Entonces lo primero que debemos hacer es crear la clase Horario antes de crear la clase Alumno.

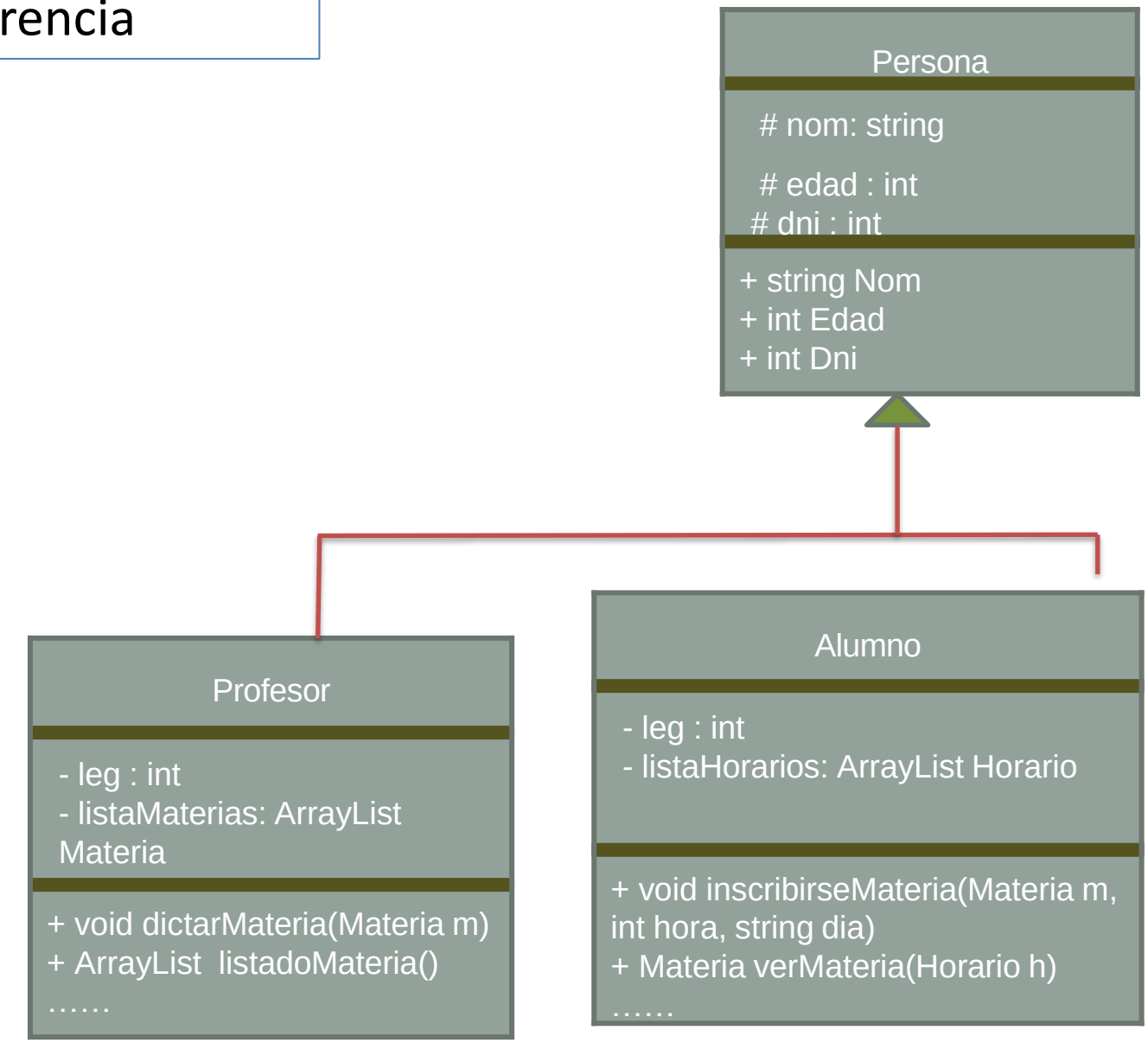


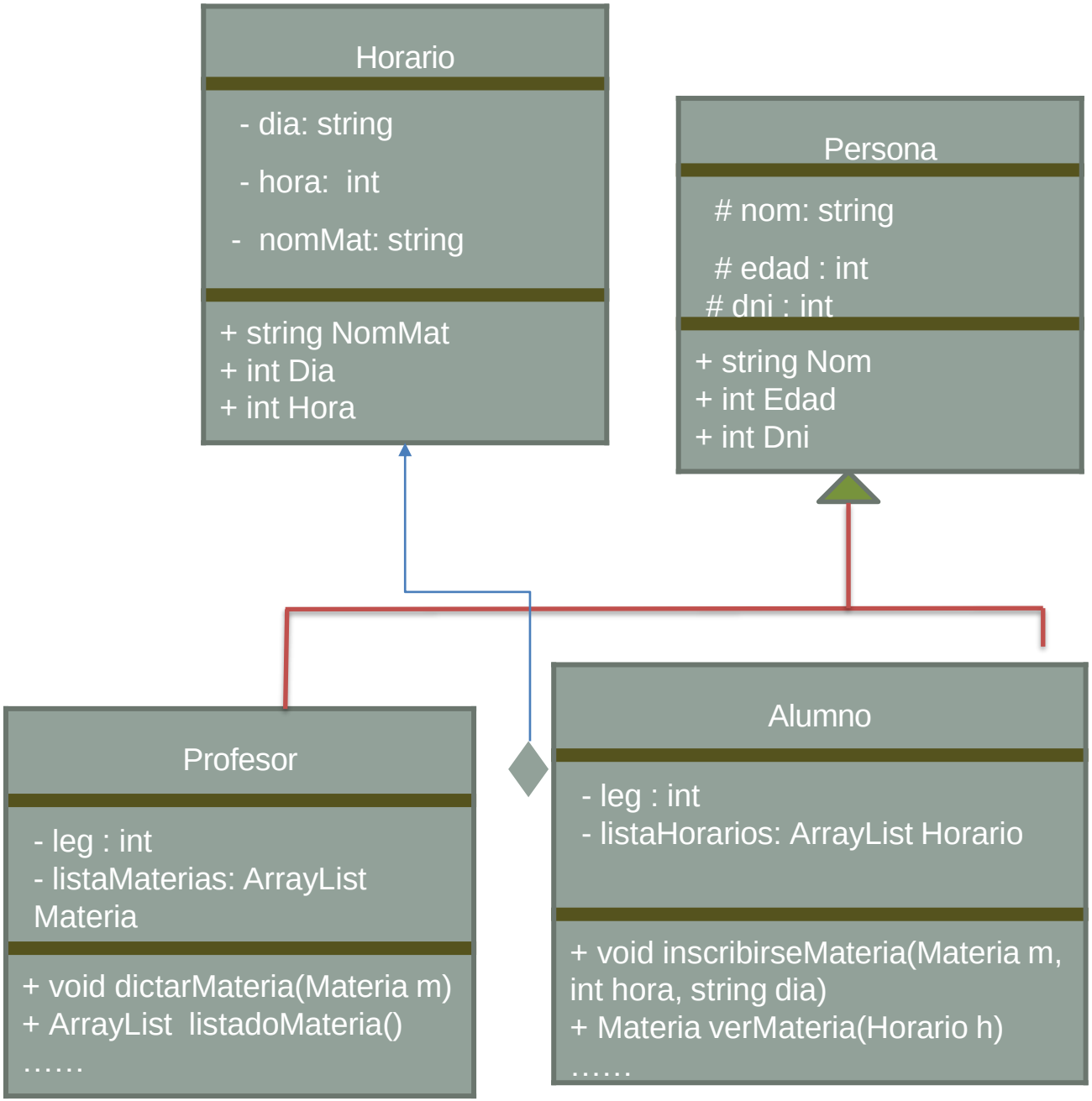
Puesta en común:

2) Implemente la clase Profesor para modelar profesores en una Universidad. Agréguele como estado lo que considere necesario para el problema. Considere que un profesor conoce la lista de materias que dicta.

¿Podemos encontrar similitudes entre las clases Alumno y Profesor?

Usamos herencia

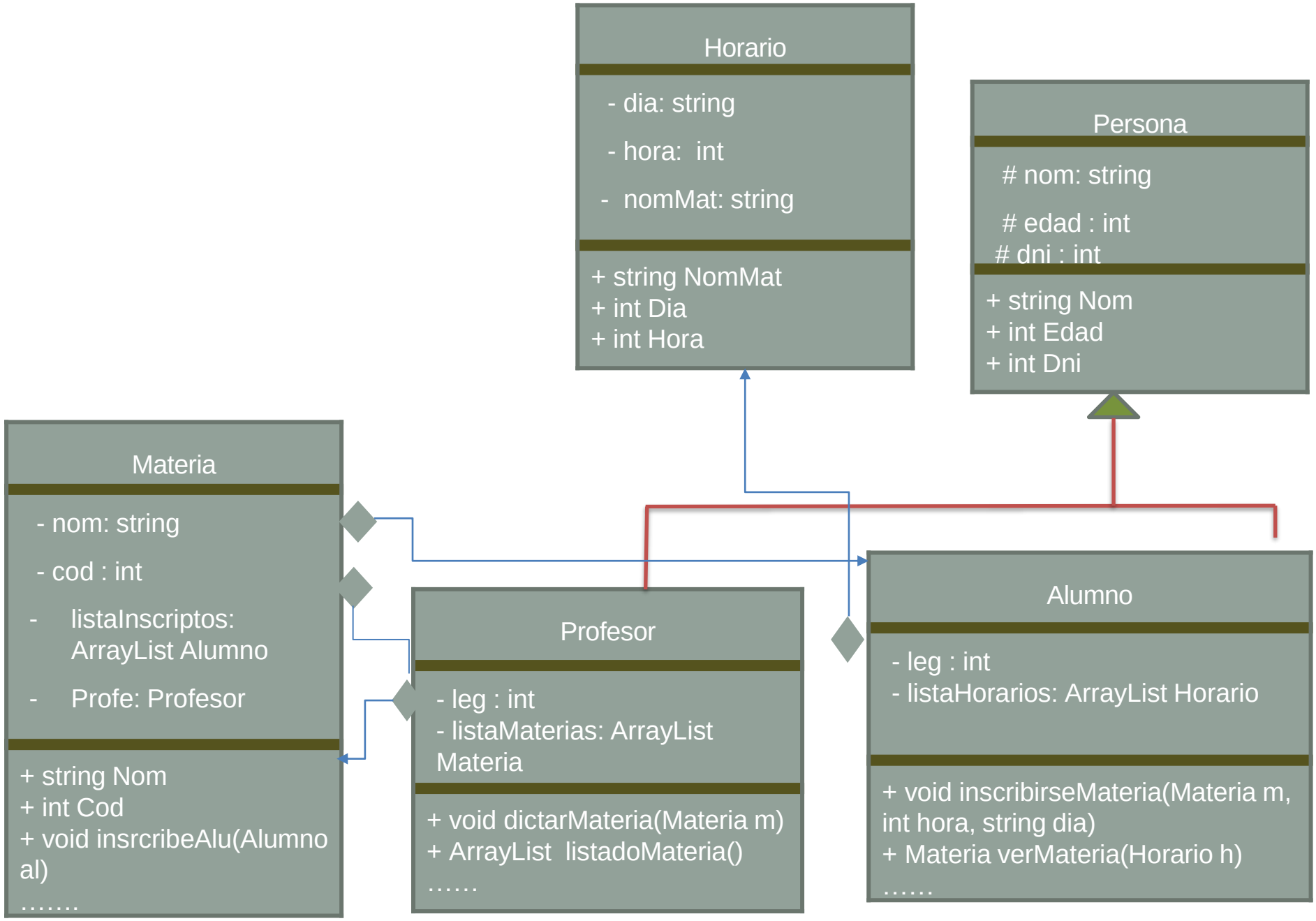


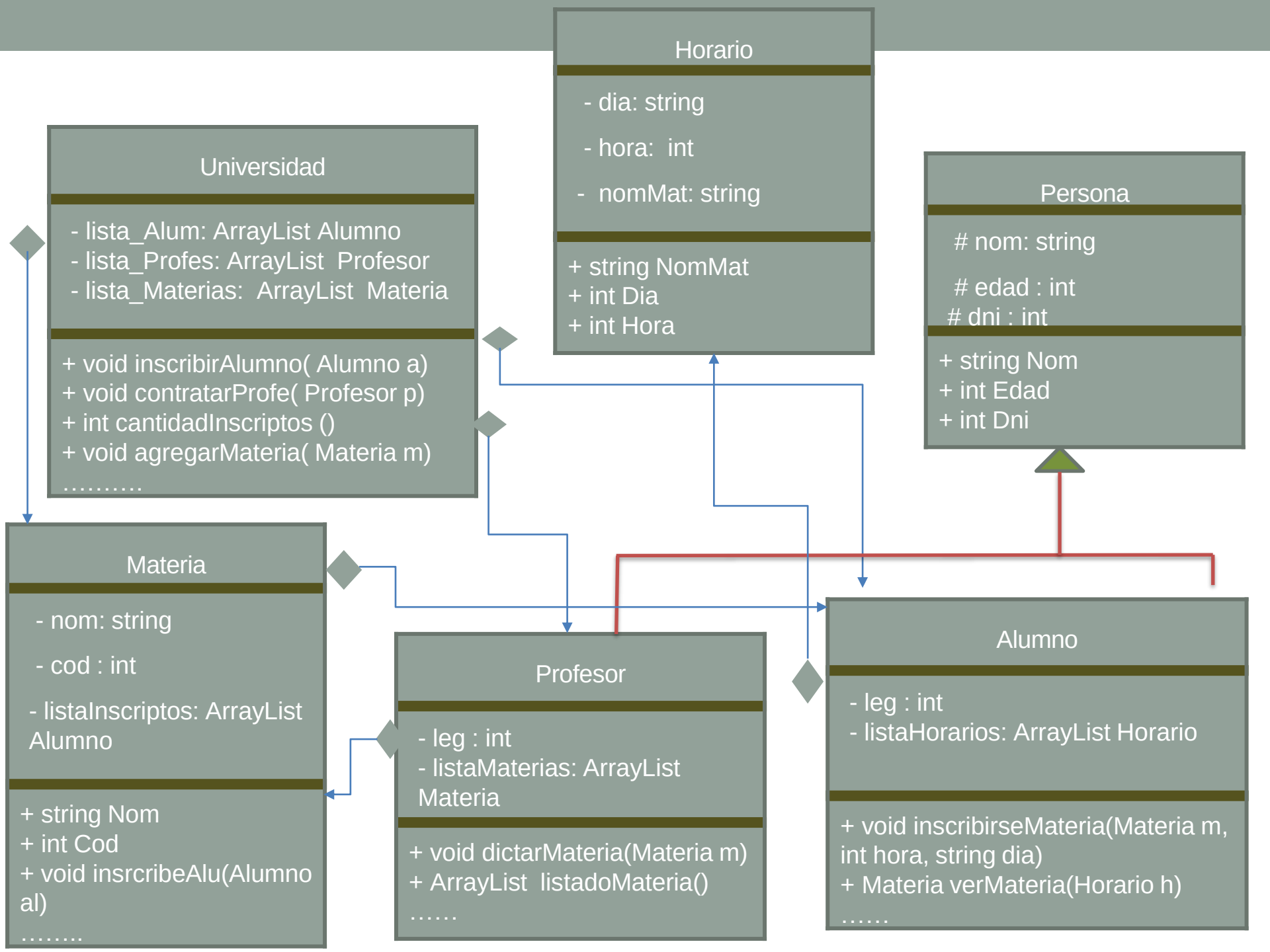

Horario
- dia: string
- hora: int
- nomMat: string
+ string NomMat
+ int Dia
+ int Hora
Persona
nom: string
edad : int
dni : int
+ string Nom
+ int Edad
+ int Dni
Profesor
- leg : int
- listaMaterias: ArrayList Materia
+ void dictarMateria(Materia m)
+ ArrayList listadoMateria()
……
Alumno
- leg : int
- listaHorarios: ArrayList Horario
+ void inscribirseMateria(Materia m, int hora, string dia)
+ Materia verMateria(Horario h)
……

Puesta en común:

3) Agregue la clase Materia que permita asociarle el profesor y la lista de los alumnos inscriptos en ella.

Finalmente utilizando las clases definidas en los ejercicios anteriores implemente la clase Universidad que permita almacenar un conjunto de profesores, un conjunto de materias y los alumnos.


Horario
- dia: string
- hora: int
- nomMat: string
+ string NomMat
+ int Dia
+ int Hora
Persona
nom: string
edad : int
dni : int
+ string Nom
+ int Edad
+ int Dni
Materia
- nom: string
- cod : int
- listaInscriptos: ArrayList Alumno
- Profe: Profesor
+ string Nom
+ int Cod
+ void insrcribeAlu(Alumno al)
…….
Profesor
- leg : int
- listaMaterias: ArrayList Materia
+ void dictarMateria(Materia m)
+ ArrayList listadoMateria()
……
Alumno
- leg : int
- listaHorarios: ArrayList Horario
+ void inscribirseMateria(Materia m, int hora, string dia)
+ Materia verMateria(Horario h)
……


Horario
- dia: string
- hora: int
- nomMat: string
+ string NomMat
+ int Dia
+ int Hora
Universidad
- lista_Alum: ArrayList Alumno
- lista_Profes: ArrayList Profesor
- lista_Materias: ArrayList Materia
+ void inscribirAlumno(Alumno a)
+ void contratarProfe(Profesor p)
+ int cantidadInscriptos ()
+ void agregarMateria(Materia m)
……
Persona
nom: string
edad : int
dni : int
+ string Nom
+ int Edad
+ int Dni
Materia
- nom: string
- cod : int
- listaInscriptos: ArrayList Alumno
+ string Nom
+ int Cod
+ void insrcribeAlu(Alumno al)
……..
Profesor
- leg : int
- listaMaterias: ArrayList Materia
+ void dictarMateria(Materia m)
+ ArrayList listadoMateria()
……
Alumno
- leg : int
- listaHorarios: ArrayList Horario
+ void inscribirseMateria(Materia m, int hora, string dia)
+ Materia verMateria(Horario h)
……