CÁTEDRA

Ingeniería de Software II

2019

Y ahora nuestro cliente quiere….

1. Poder consultar historia clínica del paciente.
2. Enviar un aviso al paciente del turno por correo.
3. Imprimir la receta para cada paciente.

Una interfaz especifica un conjunto de características públicas.
La clave es separar la especificación de funcionalidad (la interfaz) de su implementación por un clasificador como una clase o subsistema.

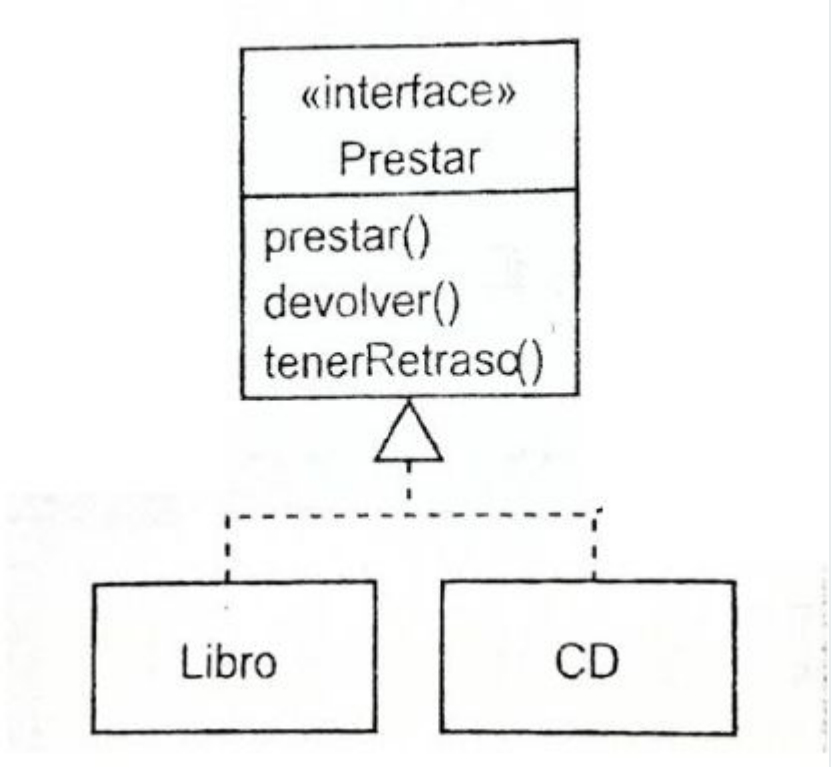
Una interfaz no se puede instanciar, simplemente declara un contrato.

Define una especificación por sus características y nunca implica ninguna implementación en particular.

Revise el diagrama de clases:

Cada libro y CD realiza la interfaz Prestar que especifica el protocolo para un elemento que se puede prestar.

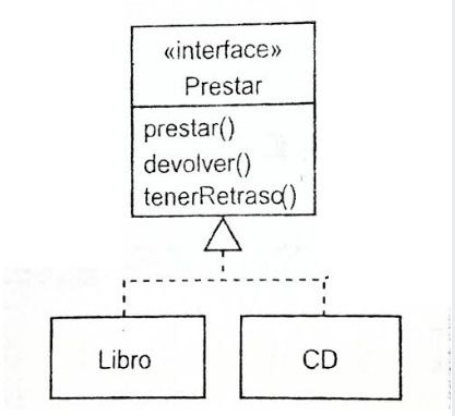
Aunque Libros y CDs pueden ser clases muy distintas, se comportan de la misma forma en cuanto al protocolo de préstamo.



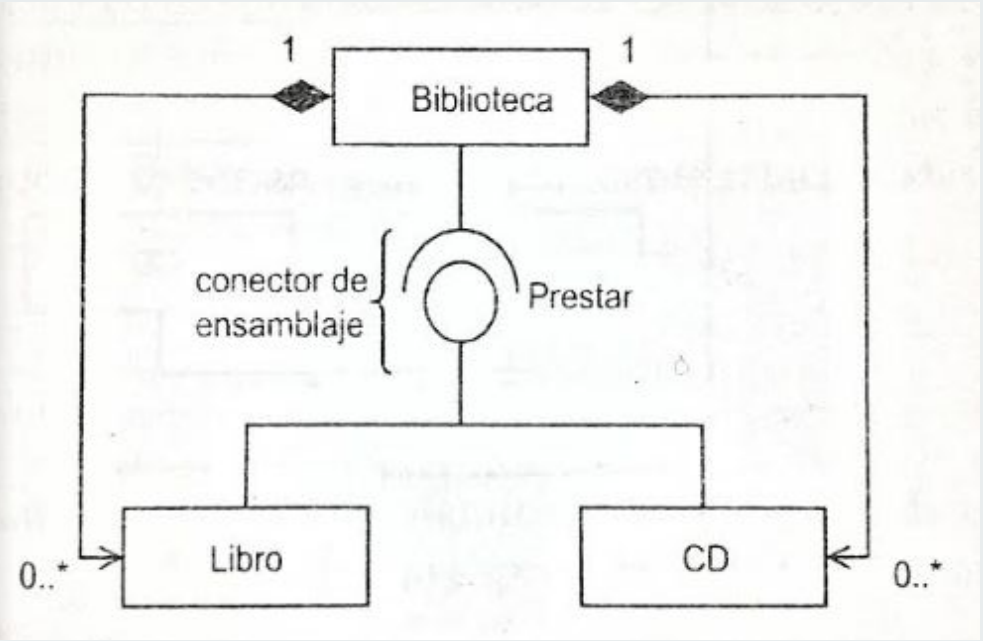
Las interfaces se nombran como clases, poniendo en mayúscula la primera letra de cada palabra.

Otra forma de nombrarlas es IPrestar o IPrestable

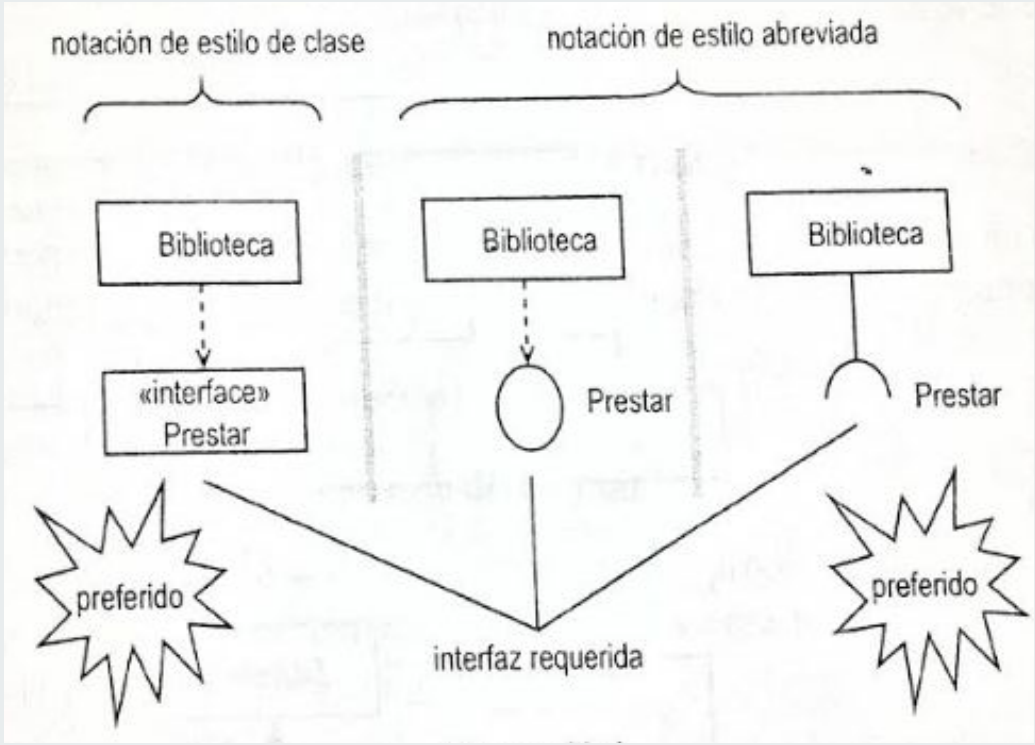
La relación se dibuja como una línea de puntos con una punta de flecha no rellena.



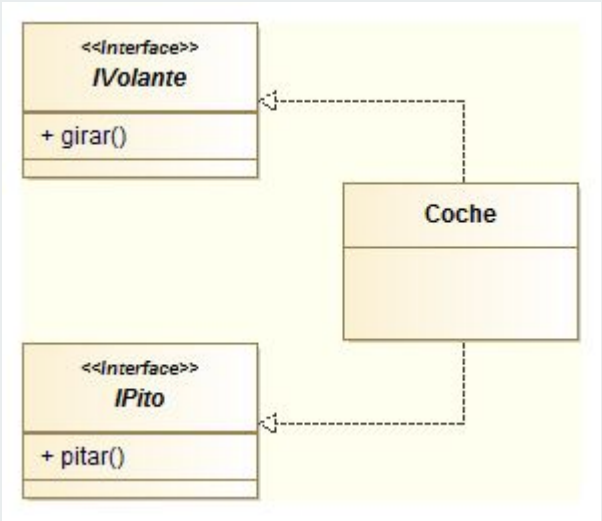
¿Qué se interpreta?



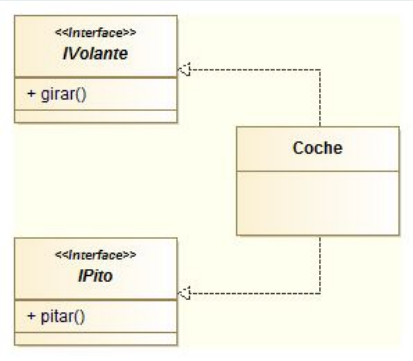
Notaciones


notación de estilo de clase
notación de estilo abreviada
Biblioteca
«interface»
Prestar
Biblioteca
Prestar
Biblioteca
Prestar
preferido
interfaz requerida
preferido

La interfaz se puede representar como:



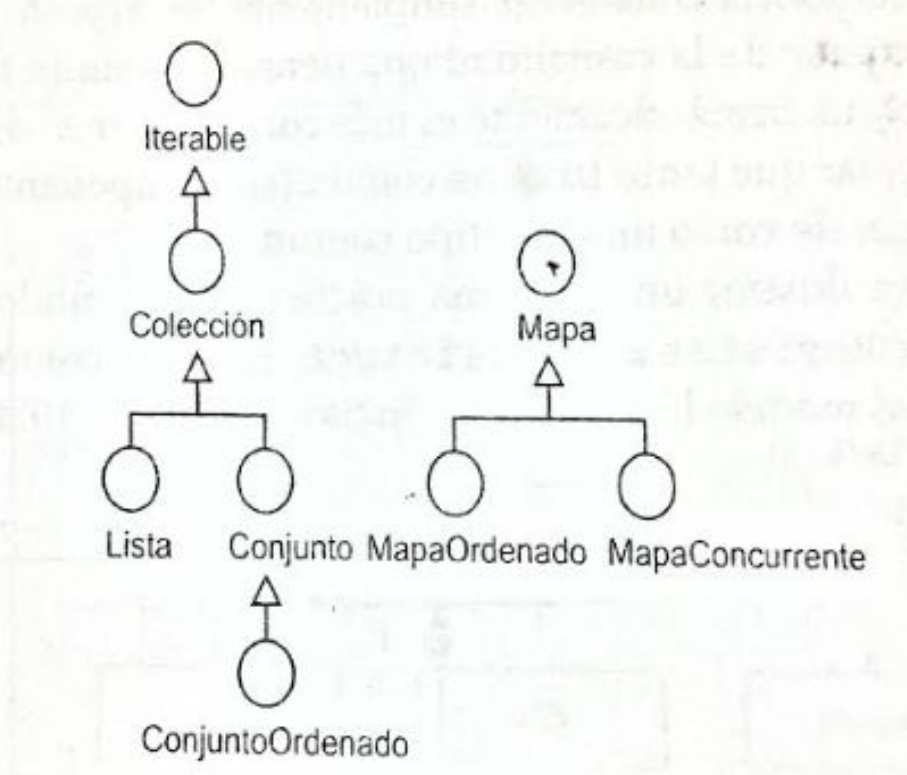
Un coche puede girar y puede pitar. Así, una clase Coche puede implementar simultáneamente el interfaz IVolante y el interfaz IPito, cuya representación podría quedar como sigue:



Un coche puede girar y puede pitar. Así, una clase Coche puede implementar simultáneamente el interfaz IVolante y el interfaz IPito, cuya representación podría quedar como sigue:

```
1  public interface IVolante {
2     public void girar();
3  }
4  public interface IPito {
5     public void pitar();
6  }
7  class Coche implements IVolante, IPito {
8     public void girar() {
9        System.out.println("¡Girando, girando!");
10    }
11    public void pitar() {
12       System.out.println("¡Pitando, pitando!");
13    }
14 }
```

En las clases de coleccion en las bibliotecas estándar de Java se pueden encontrar ejemplos de interfaces con múltiples implementaciones.



Ejercicio

Diseñe un **diagrama de clases** que modele la estructura necesaria para manejar los datos de los **encuentros** de un **torneo** de **tenis de mesa** en la **modalidad de sorteo y eliminatoria**.
Del torneo interesa conocer la **fecha del torneo**, los **encuentros celebrados** y el **ganador**.
De **cada jugador**, que **debe de conocer perfectamente las reglas**, interesa saber el **número de federado de la federación de la que es miembro**.
De cada **persona** interesa saber sus **datos básicos**: **DNI**, **nombre completo** y **fecha de nacimiento**. La clase **Fecha** se modela con tres campos (**día**, **mes** y **año**) de tipo entero. La clase **Nif** se modela con un campo de tipo entero llamado **dni** y un campo de tipo carácter llamado **letra**.
De cada **encuentro** interesa conocer los **oponentes**, el **ganador** y el **resultado final** del marcador de cada una de las **tres partidas** que se juegan a **21 puntos**.

Primero

Se procederá a **identificar las clases a partir del enunciado** y de encapsular en ellas la información relacionada. Este paso se realizará **considerando de forma aislada** unas clases de otras.

Se procede **desde las clases más triviales a las más complejas**.

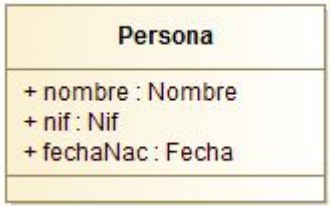

Nif
+ dni : integer
+ letra : char
Fecha
+ dia : integer
+ mes : integer
+ any : integer
Nombre
+ nombre : string
+ apellidos : string
Persona
+ nombre : Nombre
+ nif : Nif
+ fechaNac : Fecha
Jugador
+ nombre : Nombre
+ nif : Nif
+ fechaNac : Fecha
+ numFed : integer
Torneo
+ fechaTorneo : Fecha
+ encuentro : Encuentro [1..*]
+ ganador : Jugador
Encuentro
+ jugador1 : Jugador
+ jugador2 : Jugador
+ limite : integer
+ resultado : Marcador [3]
+ ganador : Jugador
Marcador
+ puntos1 : integer
+ puntos2 : integer

Segundo : **Relaciones**

- **Herencia**

Primero se abordan las **relaciones de herencia** empezando por aquellas que resulten **triviales o más evidentes**.

Persona – Jugador



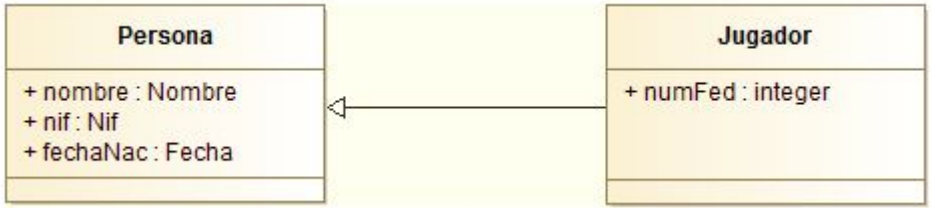
Segundo : **Relaciones**

- ## Herencia

Primero se abordan las **relaciones de herencia** empezando por aquellas que resulten **triviales o más evidentes**.

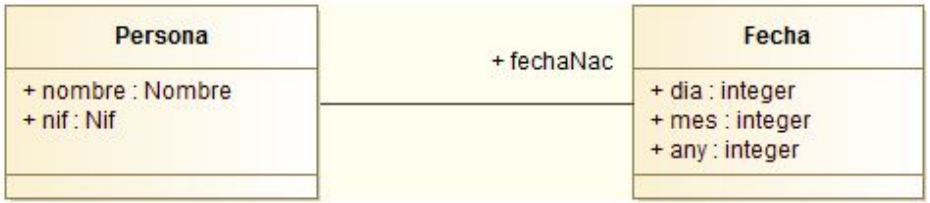
Persona – Jugador

En este caso resulta que los **atributos** de la **clase Persona** son un **subconjunto** de los de la **clase Jugador** y **semánticamente** tiene sentido decir que **la clase Jugador es una especialización de la clase Persona**.



- **Asociación (tomar de a dos clases)**

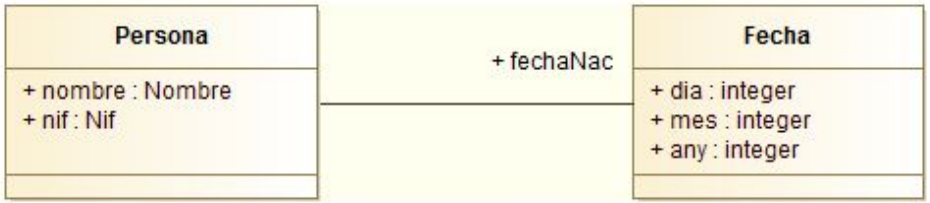
- **Asociación (tomar de a dos clases)**



- ## Asociación (tomar de a dos clases)

Una vez se han resuelto las relaciones de **herencia** le toca el turno a las relaciones de **asociación**. La clase **Persona** tiene un **atributo** de tipo **Fecha**, dicho de otra manera, **la clase Persona tiene una referencia a un objeto de la clase Fecha**.

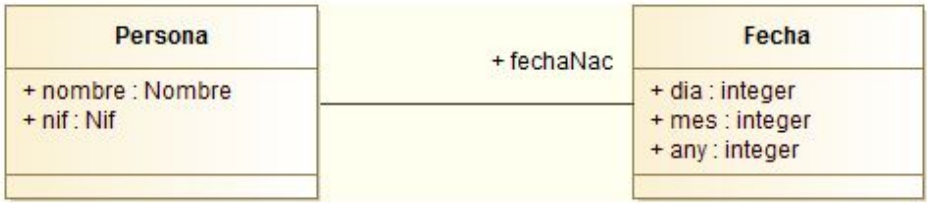
Así considerado, el atributo **fechaNac** de la clase **Persona** pasa a ser el **rol de la relación** que vincula a ambas clases. Por lo tanto, **desaparece** de la clase **Persona** y **aparece** en la **línea de vinculación junto a la clase de su tipo**.



- ## Asociación (tomar de a dos clases)

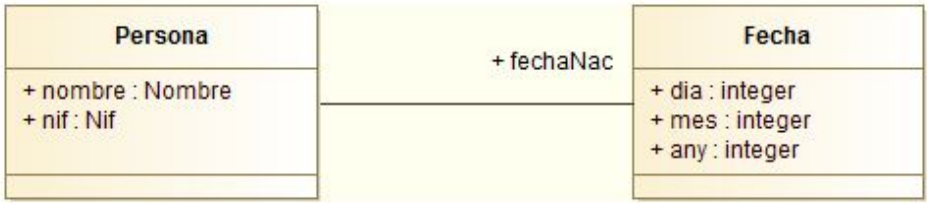
Una vez se han resuelto las relaciones de **herencia** le toca el turno a las relaciones de **asociación**.
La clase **Persona** tiene un **atributo** de tipo **Fecha**, dicho de otra manera, **la clase Persona tiene una referencia a un objeto de la clase Fecha**.

Así considerado, el atributo **fechaNac** de la clase **Persona** pasa a ser el **rol de la relación** que vincula a ambas clases. Por lo tanto, **desaparece** de la clase **Persona** y **aparece** en la **línea de vinculación junto a la clase de su tipo**.



- **Navegabilidad**

Ahora hay que abordar la **navegabilidad** tratando de ver si **desde una clase se puede ir a la otra**.

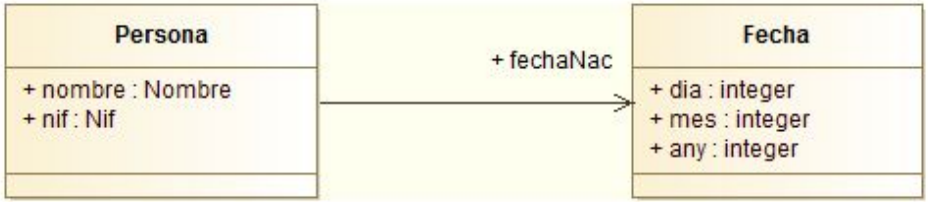


- **Navegabilidad**

Ahora hay que abordar la **navegabilidad** tratando de ver si **desde una clase se puede ir a la otra**.

Es evidente que la clase **Fecha** no tiene **información** de la clase **Persona** por lo que **la navegabilidad desde la clase Fecha no es posible**.

Sin embargo, la clase **Persona** tiene una **referencia** a la clase **Fecha** por lo que **sí es viable la navegabilidad desde la clase Persona hacia la clase Fecha**. La navegabilidad se **expresa**con **una punta de flecha abierta puesta en el lado de la clase a la que se llega**.



- **Cardinalidades**

El siguiente paso es abordar las **cardinalidades** o **multiplicidades**, es decir el **número de instancias de cada clase que intervienen en la relación**.

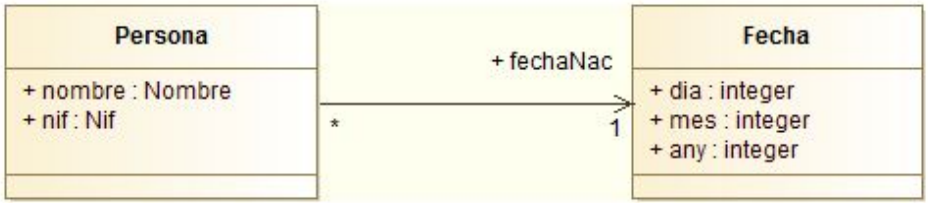
- **Cardinalidades**

El siguiente paso es abordar las **cardinalidades** o **multiplicidades**, es decir el **número de instancias de cada clase que intervienen en la relación**.

Para resolver este paso hay que preguntar:

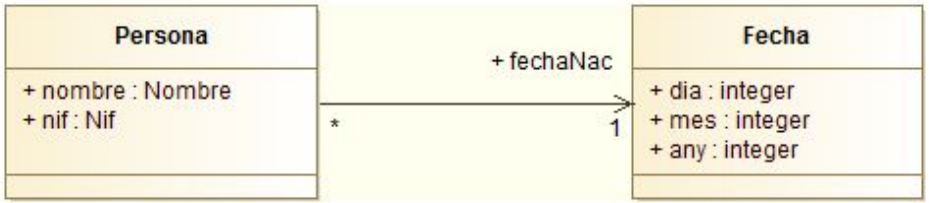
*"¿Por cada instancia de una de las dos clases cuántas instancias de la otra clase pueden en extremo intervenir como mínimo (**Cardinalidad mínima**) y como máximo (**Cardinalidad máxima**)?"*

Y luego hacer las preguntas al revés.



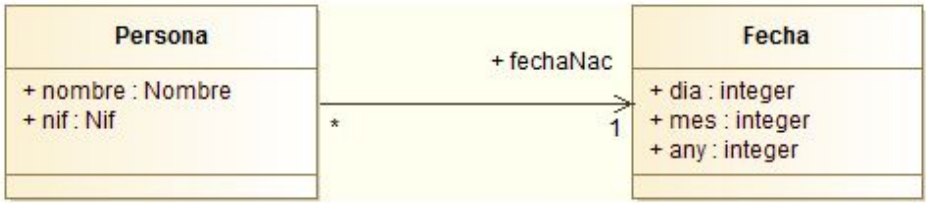
Tercer paso: TODO -PARTE

El siguiente paso consiste en considerar qué clase es la parte [**PARTE**] y qué clase es la parte [**TODO**]. Dicho de otro modo **quién contiene a quién**. En este caso la discriminación es trivial: la clase **Persona** es la parte [**TODO**] porque tiene una **referencia** a la clase **Fecha** que es la parte [**PARTE**].



- **Agregación – Composición**

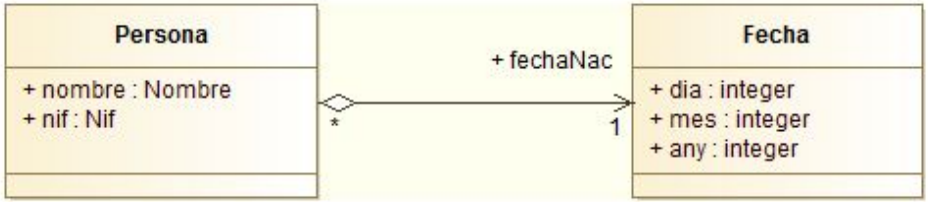
El siguiente paso consiste en considerar qué clase es la parte [**PARTE**] y qué clase es la parte [**TODO**]. Dicho de otro modo **quién contiene a quién**. En este caso la discriminación es trivial: la clase **Persona** es la parte [**TODO**] porque tiene una **referencia** a la clase **Fecha** que es la parte [**PARTE**].



- **Agregación – Composición**

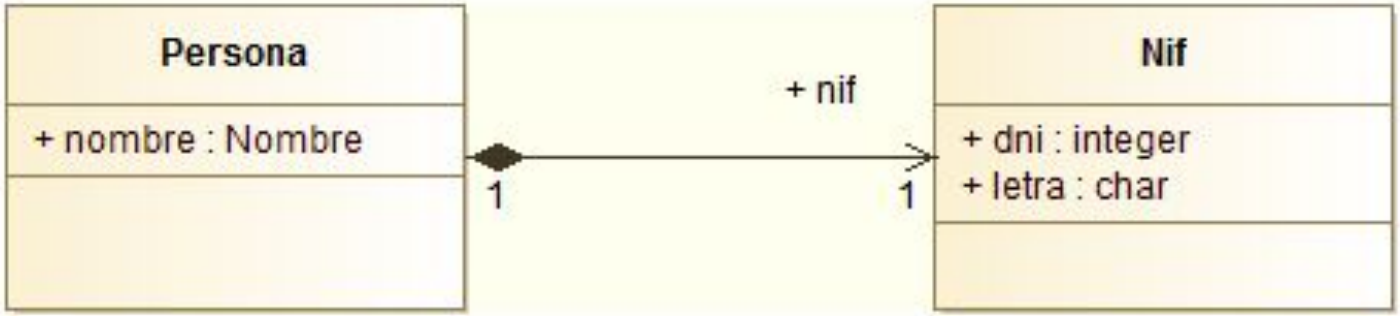
El siguiente paso consiste en considerar qué clase es la parte [**PARTE**] y qué clase es la parte [**TODO**]. Dicho de otro modo **quién contiene a quién**. En este caso la discriminación es trivial: la clase **Persona** es la parte [**TODO**] porque tiene una **referencia** a la clase **Fecha** que es la parte [**PARTE**].

Obsérvese que la parte [**TODO**] se identifica dibujando un **rombo acostado** en la línea de la relación. Obsérvese también que el se ha representado el **rombo en blanco** para identificar una relación de **agregación**.



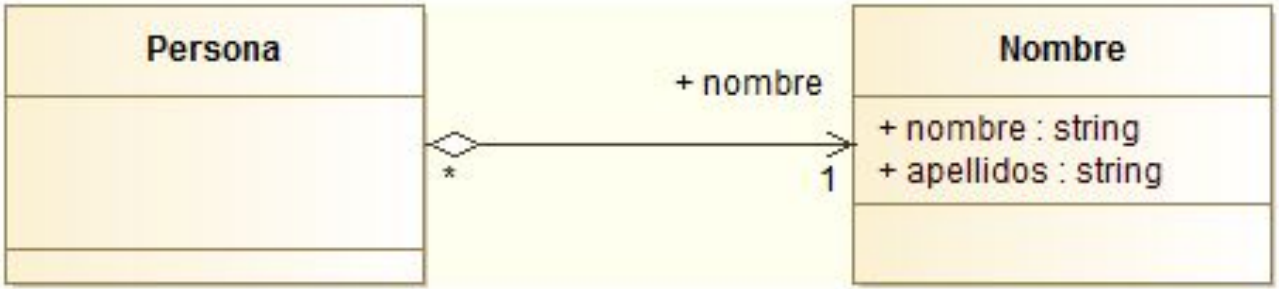
Persona – Nif (Nro de identificación fiscal)

Persona – Nif (Nro de identificación fiscal)

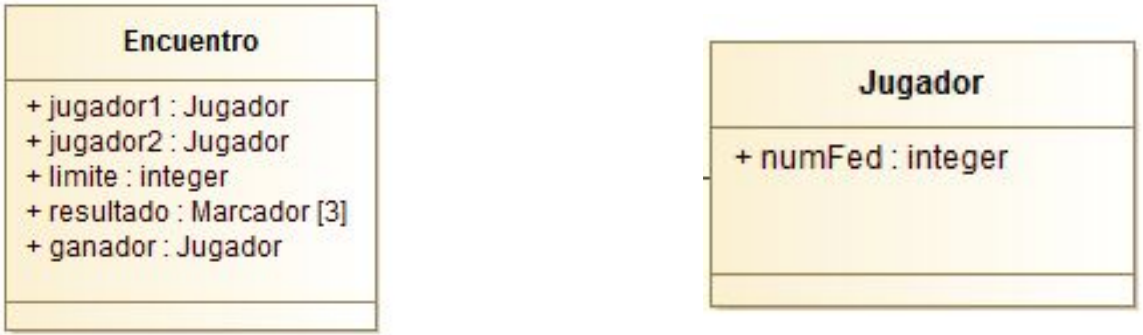


Persona – Nombre

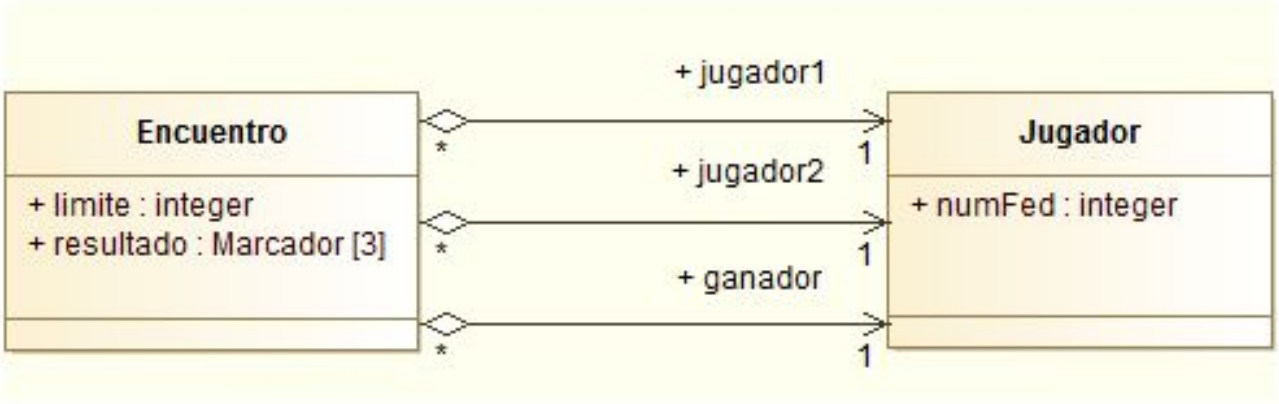
Persona – Nombre



Encuentro – Jugador



Encuentro – Jugador



Continuaremos el ejercicio en la siguiente clase

UNIDAD 4: Interfaces y componentes

Fuente:

UML 2

- “19 Interfaces y componentes”