CÁTEDRA

Ingeniería de Software II

2019

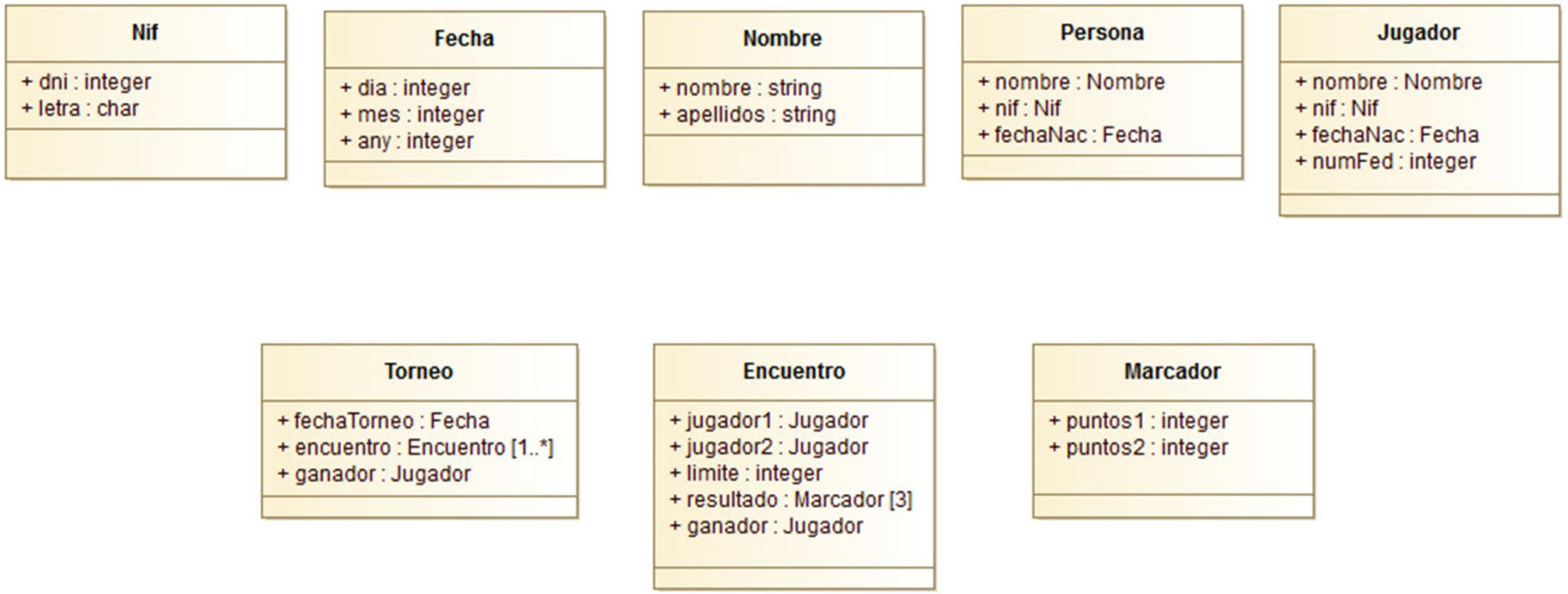
Ejercicio

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**.

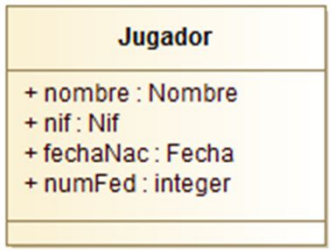

Nif
+ dni : integer
+ letra : char
Fecha
+ dia : integer
+ mes : integer
+ any : integer
Nombre
+ nombre : string
+ apellidos : string
Persona
+ nombre : Nombre
+ nif : Nif
+ fechaNac : Fecha
Jugador
+ nombre : Nombre
+ nif : Nif
+ fechaNac : Fecha
+ numFed : integer
Torneo
+ fechaTorneo : Fecha
+ encuentro : 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**.

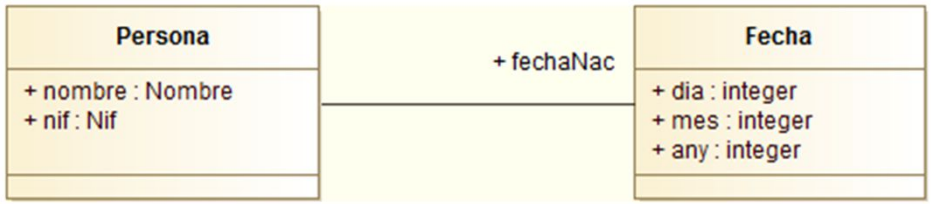
- ## Asociación (tomar de a dos clases)

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**.

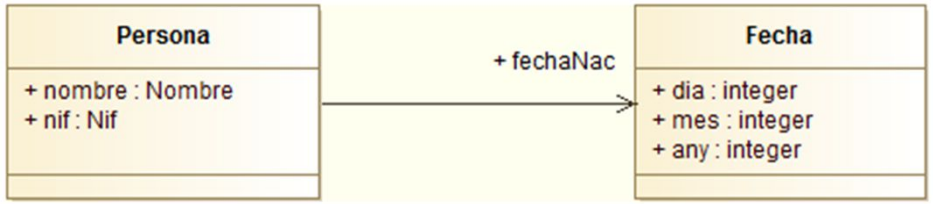


- **Navegabilidad**

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

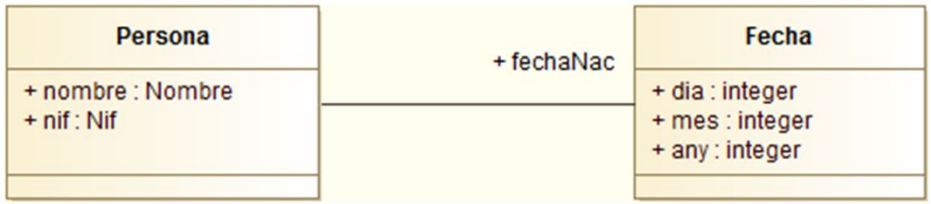
Es evidente que la clase **Fecha** no tiene **información** de la clase **Persona** por lo que **la navegabilidad desde la clase Fecha no es posible**.

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]**.

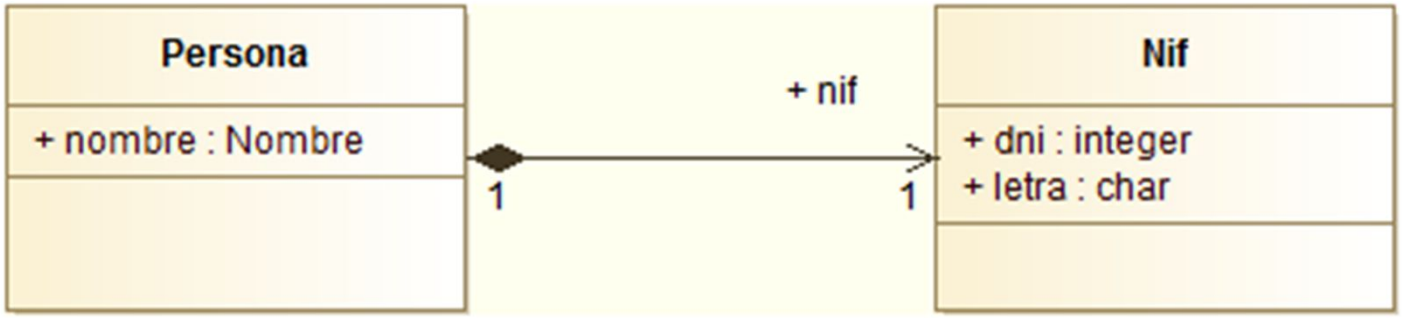
- **Agregación – Composición**

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**.

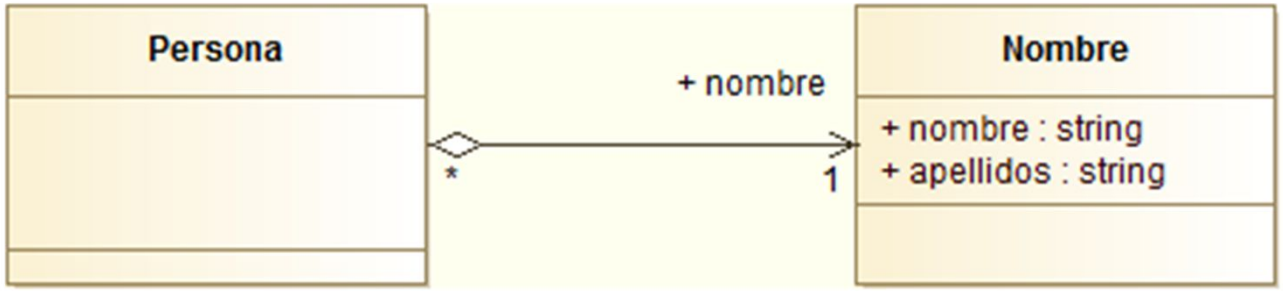
Persona – Nif (Nro de identificación fiscal)

Persona – Nif (Nro de identificación fiscal)



Persona – Nombre

Persona – Nombre

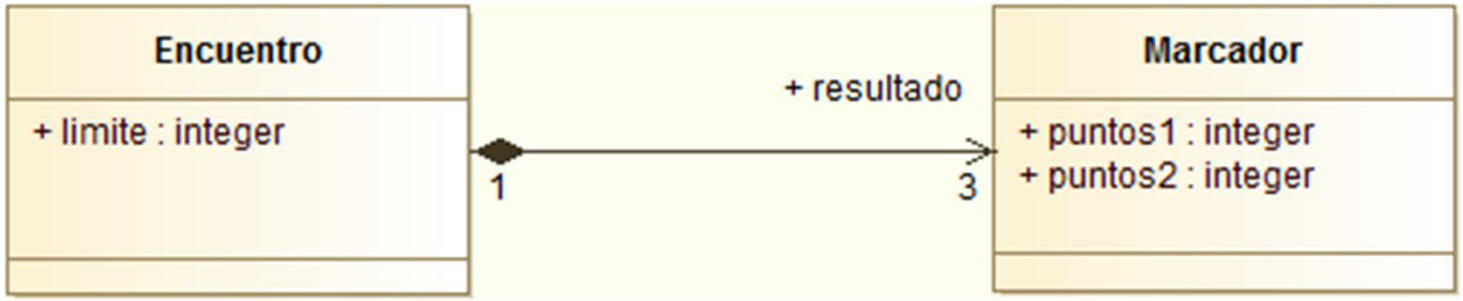


Encuentro – Jugador

Encuentro – Jugador

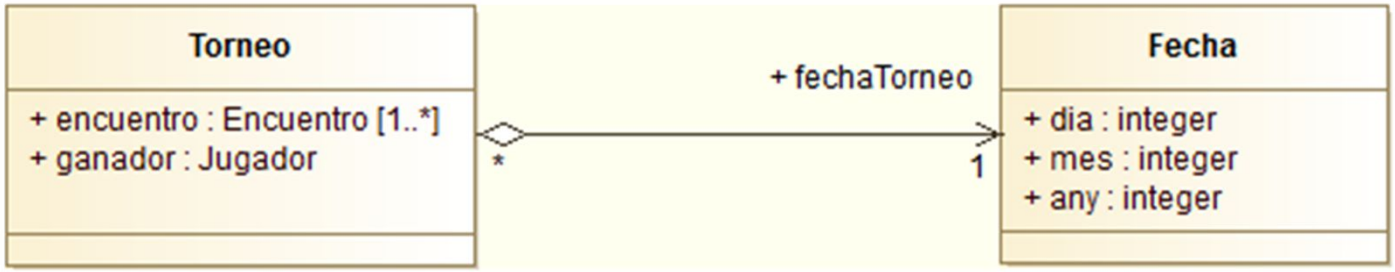
Encuentro – Marcador

Encuentro – Marcador



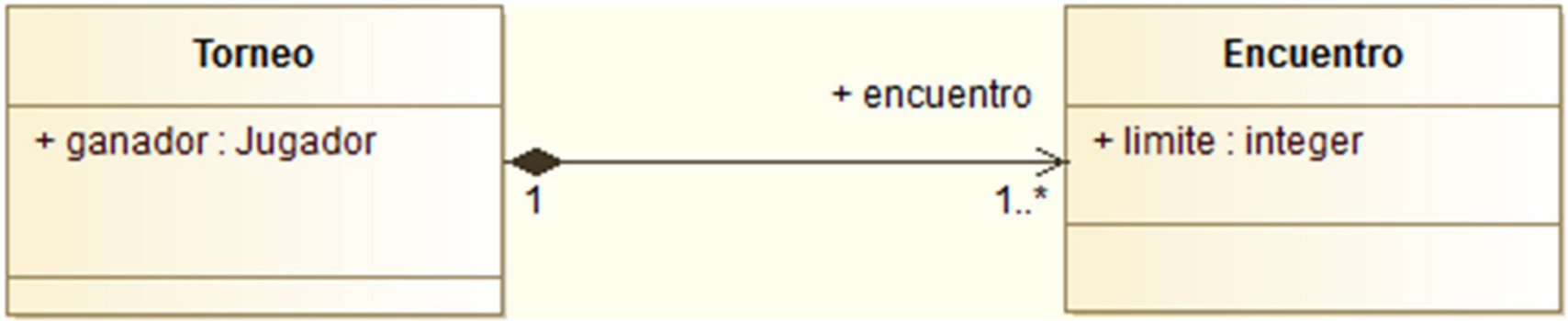
Torneo – Fecha

Torneo – Fecha



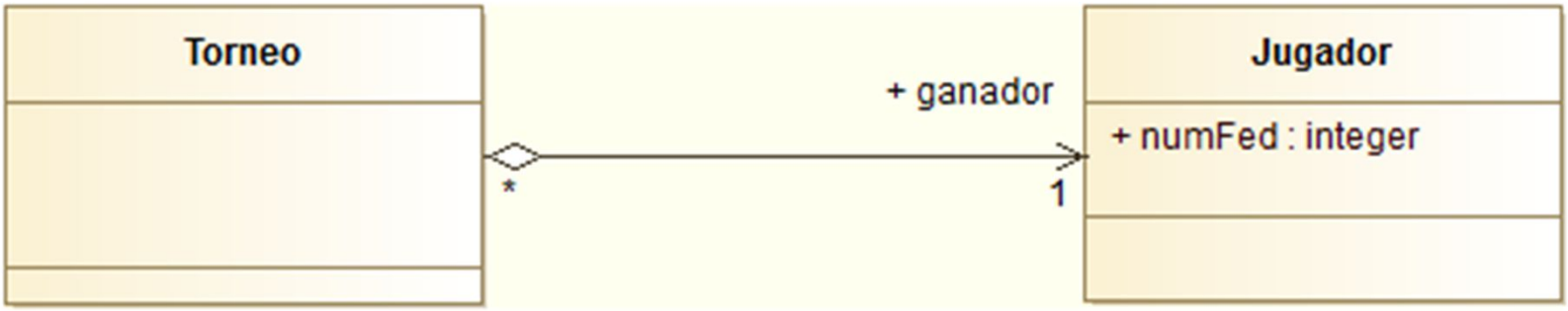
Torneo – Encuentro

Torneo – Encuentro



Torneo – Jugador

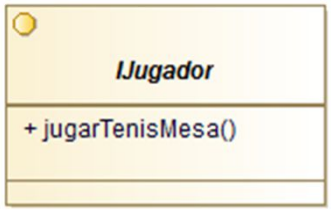
Torneo – Jugador



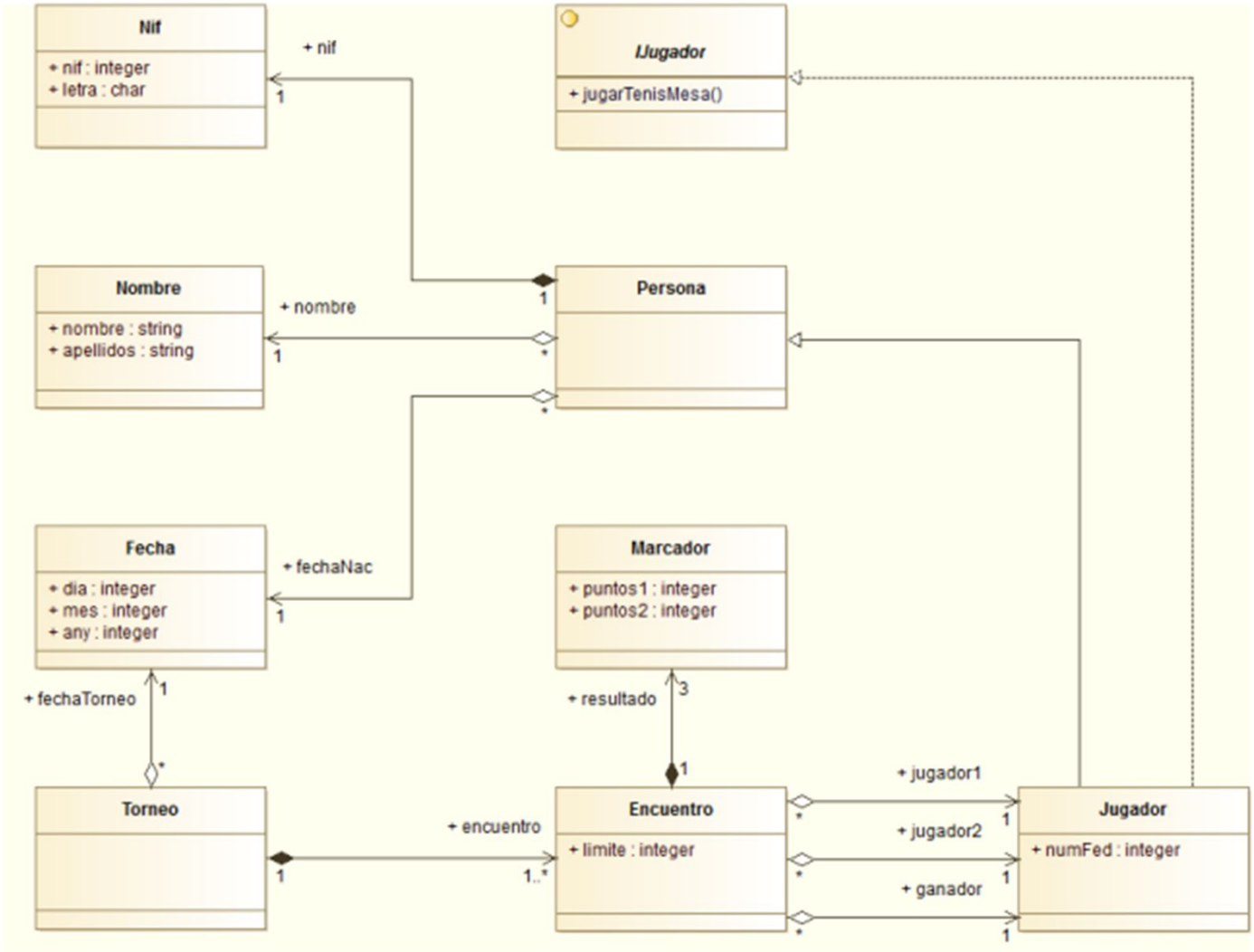
- **Interfaz IJugador**

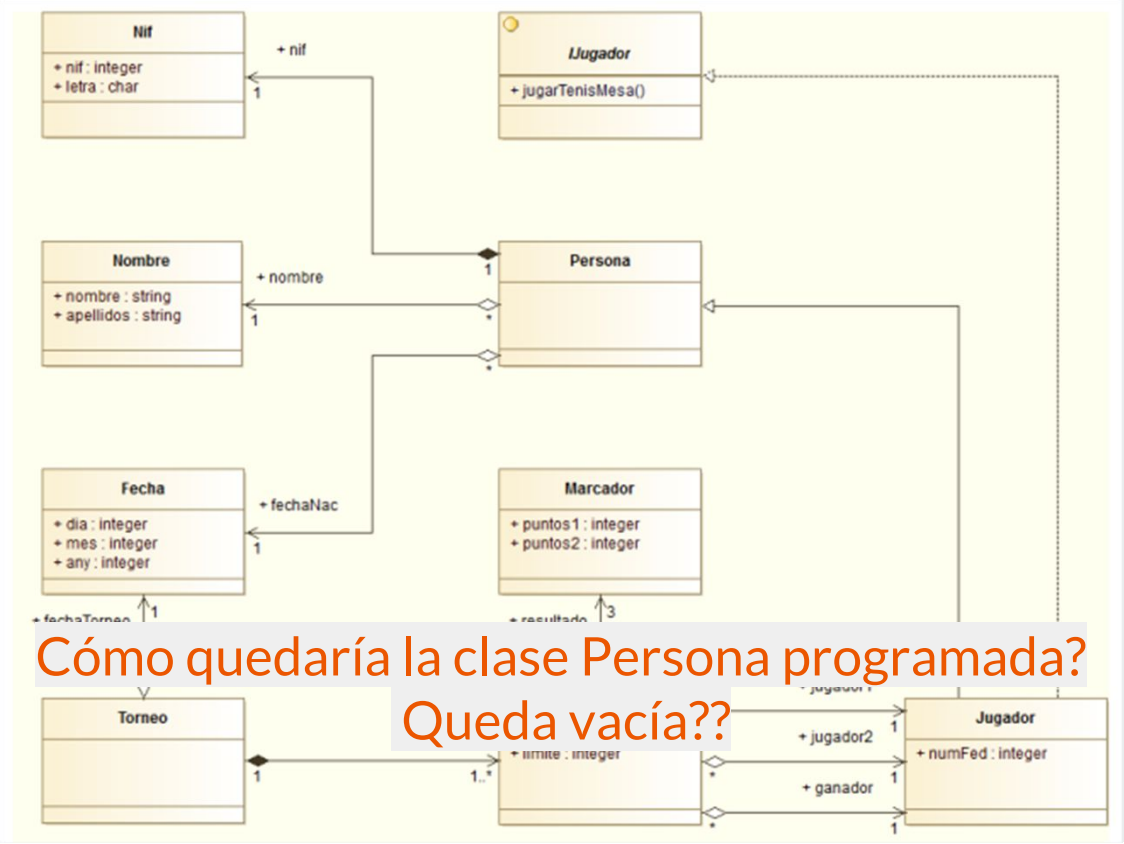
Si se conviene en que la **capacidad de jugar al tenis de mesa** viene proporcionada por el contenido de un determinado método, toda **clase** que represente a una persona que sabe jugar a este deporte **incorporará este método en su código**.

Sin embargo, ¿**Cómo reconocer a un jugador de tenis de mesa sin verlo jugar**? La respuesta viene a través de los interfaces. Un **interfaz** es como un **título** que **faculta a su poseedor** en una determinada **habilidad**. Así se reconoce a un jugador por su título, como se conoce a un médico por su título universitario, un extintor eficaz por su certificado de industria, la reparación de un coche por su factura, etc.



Jugador – IJugador


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

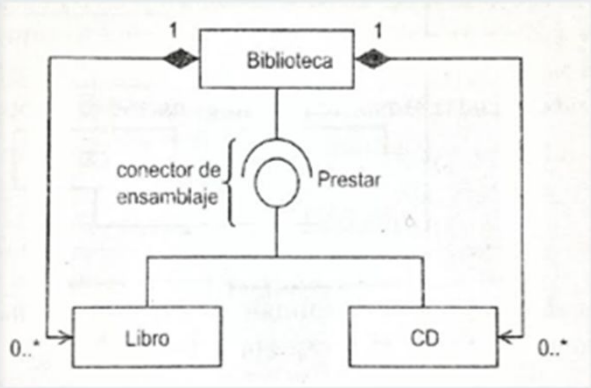

Nif
+ nif : integer
+ letra : char
+ nif
1
IJugador
+ jugarTenisMesa()
Nombre
+ nombre : string
+ apellidos : string
+ nombre
1
*
Persona
1
*
Fecha
+ dia : integer
+ mes : integer
+ any : integer
+ fechaNac
1
Marcador
+ puntos1 : integer
+ puntos2 : integer
3
Cómo quedaría la clase Persona programada?
Queda vacía??
Torneo
1
1..*
+ jugador2
+ ganador
1
*
Jugador
+ numFed : integer

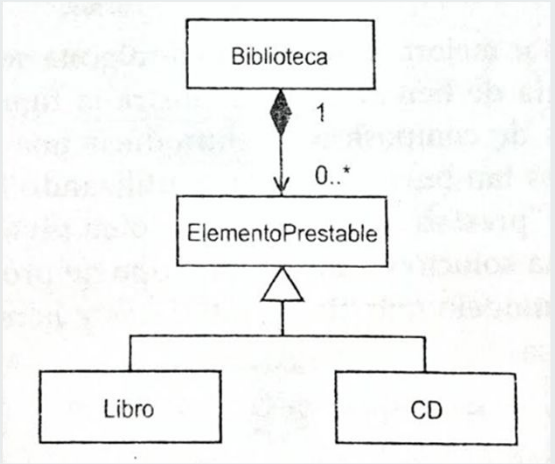
Volviendo al ejemplo de la biblioteca…

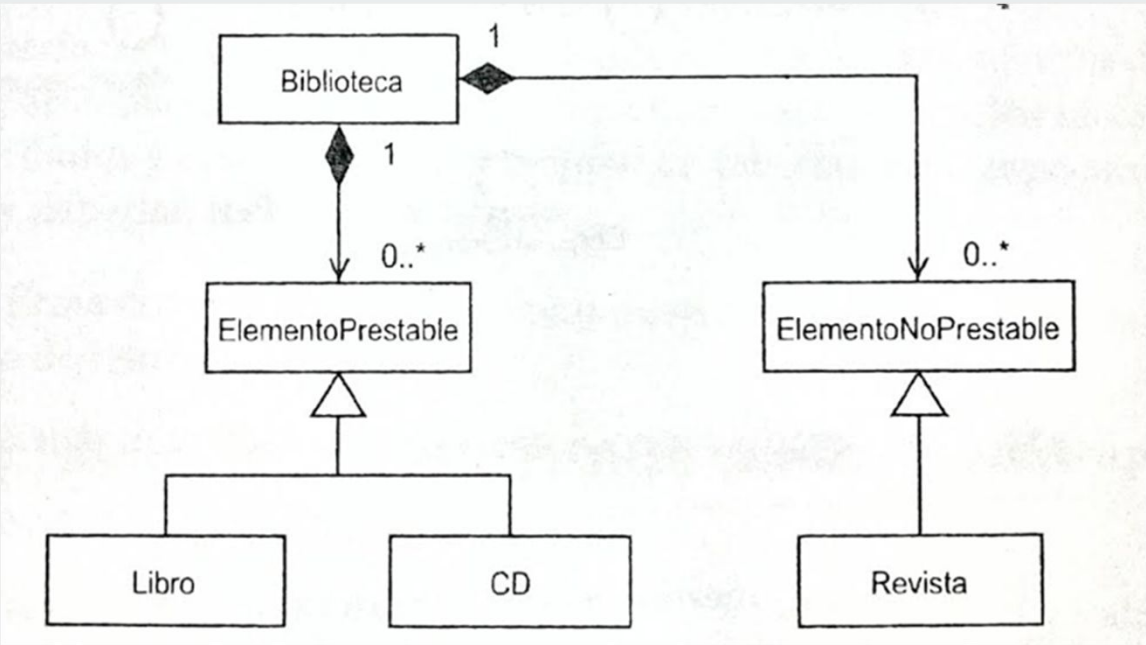
Libro realiza el contrato especificado por ElementoPrestable?

Libro es un elemento ElementoPrestable?

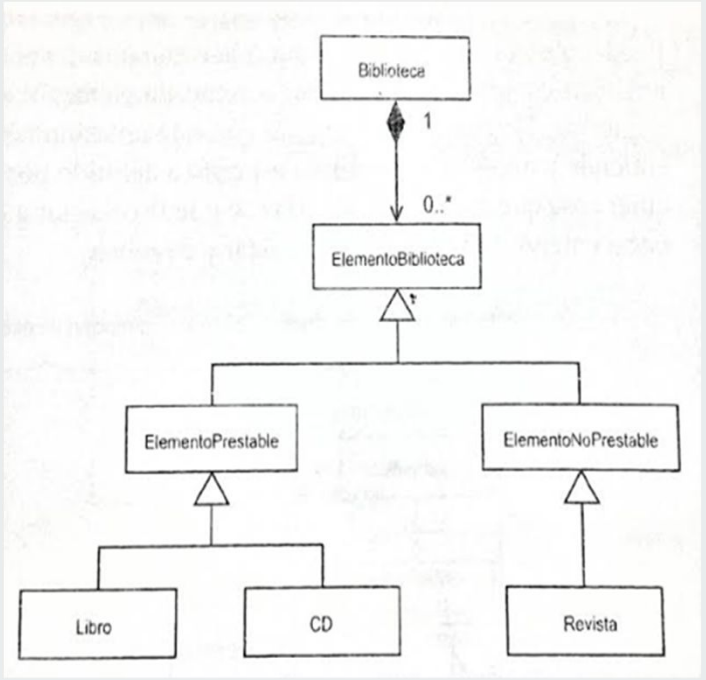
¿¿¿Herencia o realización de interfaz???






Biblioteca
1
1
0..*
0..*
ElementoPrestable
ElementoNoPrestable
Libro
CD
Revista

Añadimos un nivel mas de jerarquía de herencia, librándonos de una de las relaciones de composición al introducir Elemento Biblioteca.
Podemos proponer una solución mas elegante...



. Todo elemento en la biblioteca es un elemento biblioteca

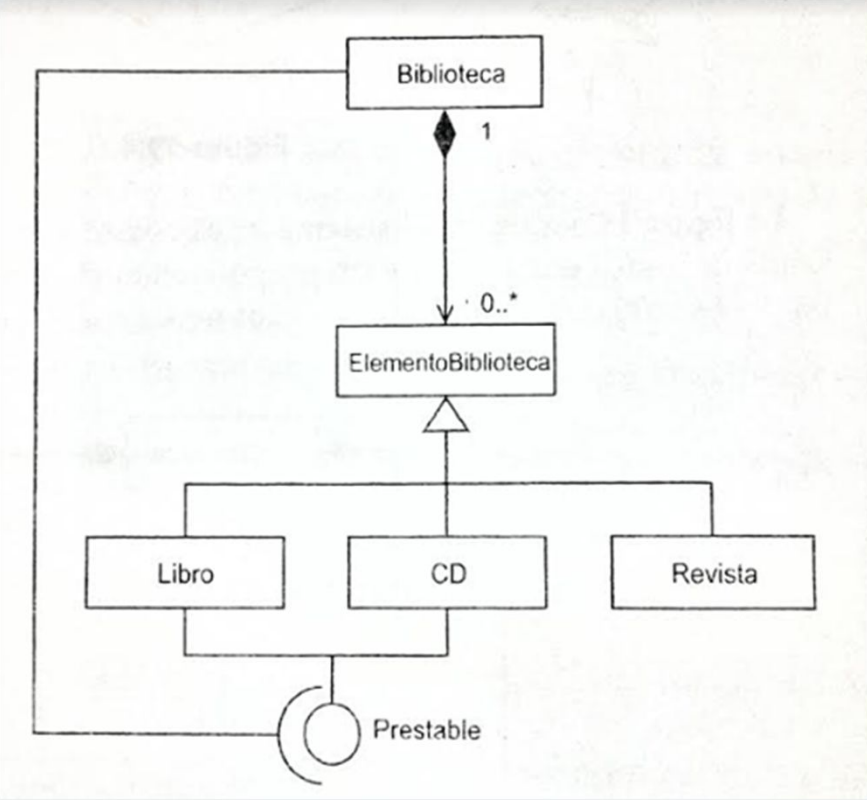
. Hemos resuelto la noción de prestabilidad en una interfaz aparte, Prestable, que podemos aplicar a Elementos biblioteca según sea necesario

. Menos clases que la solución anterior.

.Poseemos menos relaciones de composición

. Jerarquía de herencia con solamente dos niveles

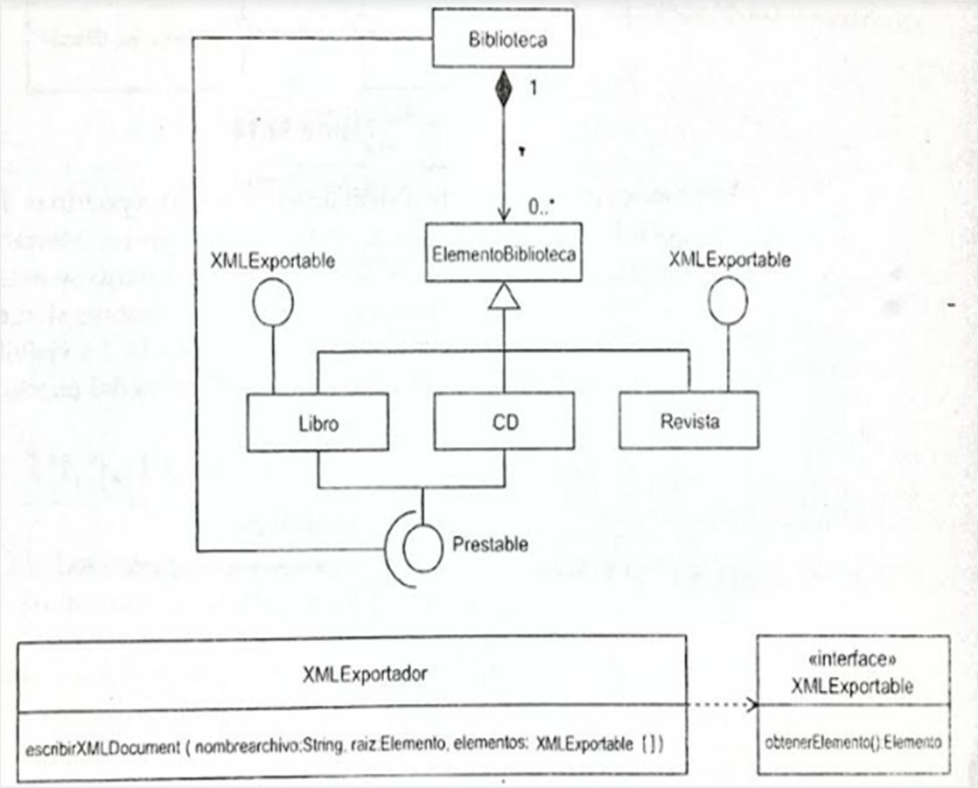
. Tenemos menos relaciones de herencia.



Supongamos que queremos exportar detalles de libros y revistas (no CD)

Agregamos

- XML Exportador que realiza la exportación.
- XML Exportable que define el protocolo
- Se relacionan las clases Libro y Revista a esta nueva interfaz.



Las interfaces son la clave para el desarrollo basado en componentes

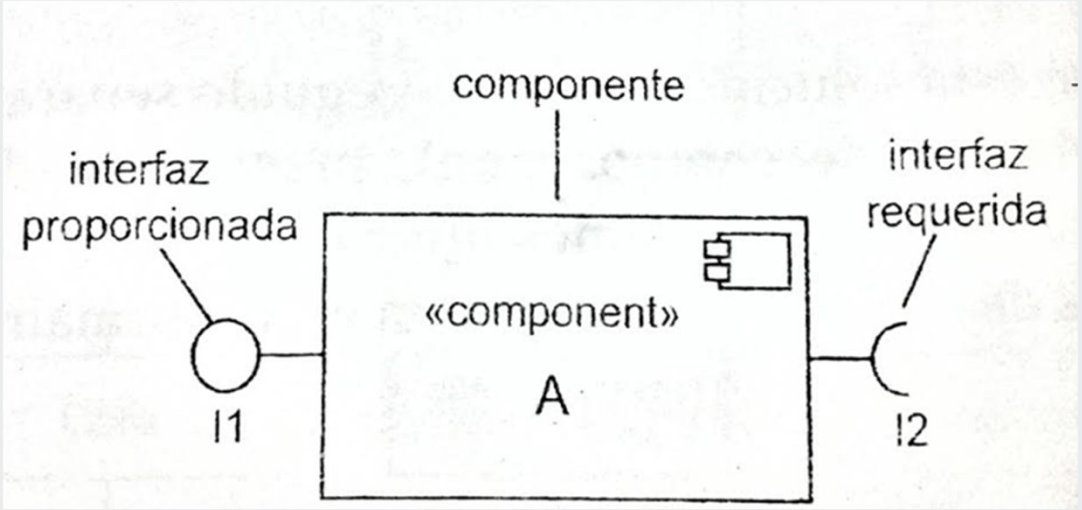
Si desea crear software flexible basado en componentes para lo que puede incorporar nuevas implementaciones, debe diseñar con interfaces.

Puesto que una interfaz solamente especifica un contrato, permite cualquier número de implementaciones específicas, siempre y cuando cada uno se ajuste a ese contrato.

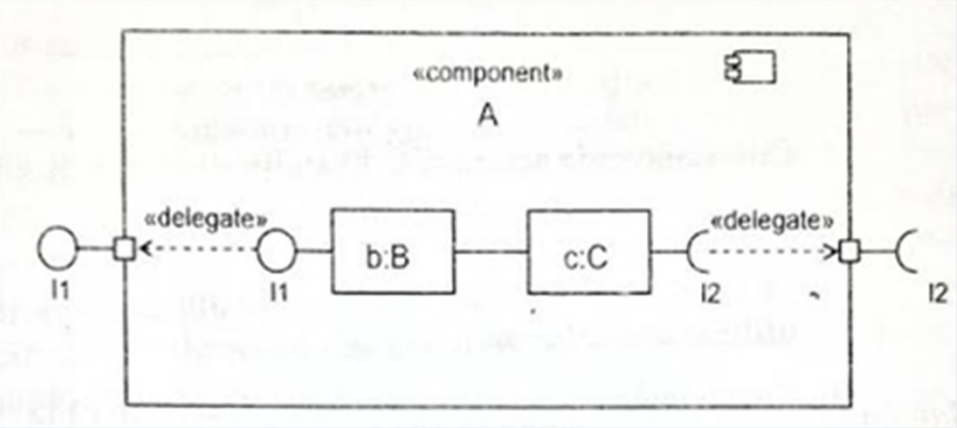
Un componente representa una parte modular de un sistema que encapsula sus contenidos y cuya manifestación se reemplaza dentro de su entorno.

Actúa como una caja negra cuyo comportamiento externo está completamente definido por sus interfaces proporcionadas y requeridas.

Debido a esto, un componente se puede reemplazar por otro que soporte el mismo protocolo.


componente
interfaz
proporcionada
interfaz
requerida
«component»
A
I1
I2

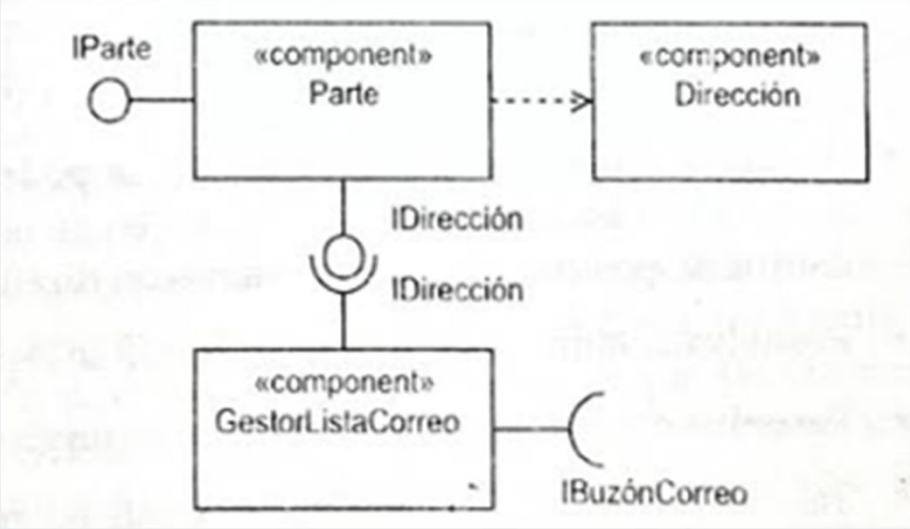
- El componente A proporciona la interfaz I1
- Requiere la interfaz I2
- Encapsula dos partes de tipo b y c
- Delega a éstas el comportamiento especificado



El componente parte proporciona dos interfaces.

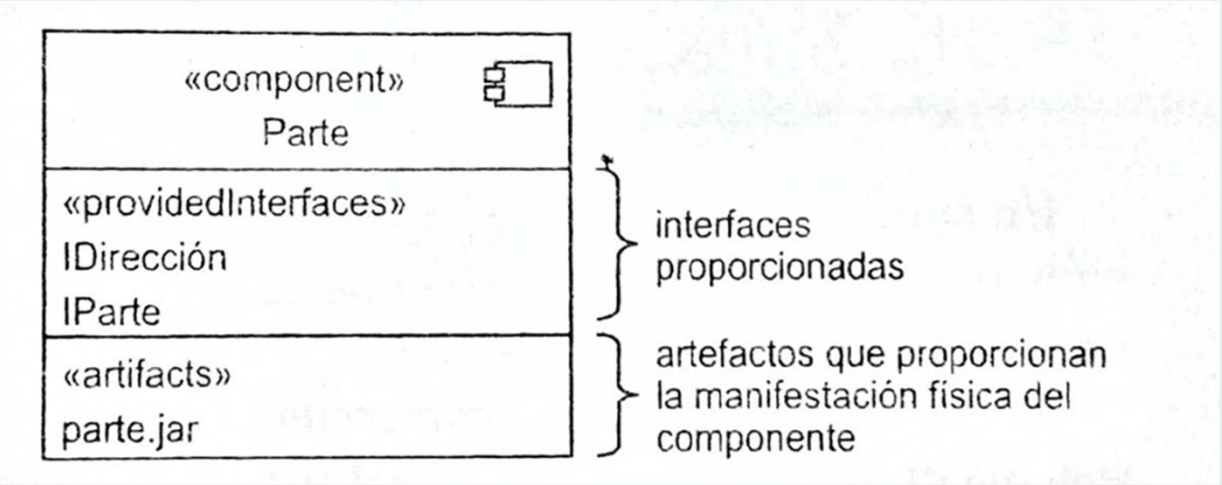
El componente Gestor Lista correo requiere dos interfaces.

Existe un conector de ensamblaje entre el componente Parto y el componente GestorListaCorreo.



Los componentes pueden mostrarse como caja blanca

Esta vista presenta los detalles internos del componente



Ahora uds!!

En equipo pensar en una actividad sencilla que pueda dividir en componentes y relacionarlos mediante interfaces

Diseño Componentes-Interfaz

Primero se definen los componentes - las grandes piezas del sistema - y después las interfaces correspondientes

Una vez que los componentes y las interfaces se han definido es posible dividir la implementación del sistema entre los participantes organizandolos en varios grupos los desarrolladores son libres de implementar los componentes.

Diseño a partir de clases

Utilizando las clases dominio y el método a partir de las clases se tiende más a la resolución del problema

En caso de que la complejidad de la solución se incremente o se identifique un grupo de clases que pueda depurarse y reutilizarse con más facilidad si se les encapsula en componentes.

UNIDAD 4: Interfaces y componentes

Fuente:

UML 2

- “19 Interfaces y componentes”

Manual de UML, Paul Kimmel

- “8 Modelado de componentes”