Evolución de la POO

- Años 60 **Simula**
  - Resolución de problemas de simulación
  - Ole-Johan Dahl & Krysten Nygaard (Noruega)
- Años 70 **Smalltalk**
  - Entorno de programación entendible por “novatos”
  - Alan Kay (Xerox PARC, Palo Alto, California)
- Años 80 **C++**
  - Extensión de C
  - Bjarne Stroustroup (AT&T Bell Labs)
- Años 90 **Java**
  - *“Write once, run everywhere”*
  - Sun Microsystems

Análisis y Diseño Orientado a Objetos

- Es un enfoque de la **ingeniería de software** que permite modelar un sistema como un grupo de objetos que interactúan entre sí.
- Este enfoque representa un dominio en términos de conceptos compuestos por sustantivos y verbos, clasificados de acuerdo a su dependencia funcional.

Desarrollo de Software

- El desarrollo de software implica cuatro **actividades** básicas.
  - **Establecer requisitos**.
  - **Crear un diseño.**
  - **Implementar código**.
  - **Probar la implementación.**
- Estas actividades no son estrictamente lineales, pueden solaparse e interactuar entre sí.

Establecer Requisitos

- Especificar las tareas de un programa
  - **Qué hace**, no cómo lo hace
- Un conjunto de requisitos debe ser mejorado y extendido.
- Es difícil establecer requisitos detallados, inambigüos y completos.
- Una atención cuidadosa a los requisitos puede ahorrar tiempo y dinero en el proyecto completo.

Crear un Diseño

- Específica **cómo** un programa logrará sus requisitos.
- Específica como la solución puede ser descompuesta en partes más simples y que hará cada una.
- Un diseño orientado a objetos determina las clases necesarias y cómo interactúan.
- El diseño a bajo nivel incluye como los métodos lograrán sus tareas.

Implementar código

- La implementación es el proceso de pasar un diseño a código.
- Los programadores novatos piensan que escribir código es la parte esencial del desarrollo de software, pero debe ser el paso menos creativo.
- La implementación debe centrarse en detalles de código, incluyendo guías de estilo y documentación.

Pruebas

- Las pruebas intentan asegurar que el programa resuelva el problema bajo las restricciones especificadas en los requisitos.
- Un programa debería ser probado para encontrar los errores.
- La depuración es el proceso que determina la causa de los problemas.

Ventajas del Diseño Orientado a Objetos

- **Robustez**
- **Adaptabilidad**
- **Reusabilidad**

Robustez

- Un software es **robusto**, si es capaz de manejar entradas inesperadas que no están explícitamente definidas para su aplicación.
- Por ejemplo, si un programa está esperando un número entero positivo (que representa el precio de un artículo) y en su lugar se da un número entero negativo, entonces el programa debe ser capaz de recuperarse correctamente de este error.

Adaptabilidad

- Software capaz de evolucionar con el tiempo en respuesta a las condiciones cambiantes de su entorno.
- Un concepto relacionado a la adaptación de software es la **portabilidad**, que es la habilidad de un software para ejecutarse con cambios mínimos en diferente hardware y diferentes sistemas operativos.

Reusabilidad

- El software es **reutilizable**, es decir, el mismo código debe ser útil como un componente en diferentes sistemas de diversas aplicaciones.
- Reduce **tiempo y costo** en el desarrollo de proyectos complejos.

Principios del Diseño Orientado a Objetos

- **Abstracción**
- **Encapsulación**
- **Modularidad**

Abstracción

- La noción de **abstracción** consiste en descomponer un sistema complejo hasta sus partes fundamentales y describir estas partes (es decir, nombrarlas y explicar su funcionalidad) en un lenguaje sencillo y preciso.

- Aplicar el **paradigma de abstracción** para el diseño de estructuras de datos da lugar a **tipos de datos abstractos** (TDA).

Abstracción

- Un TDA es un modelo matemático de una estructura de datos que específica el tipo de los datos almacenados, las operaciones soportadas en ellos, y los tipos de los parámetros de las operaciones.

- Un TDA especifica lo que cada operación hace, pero no cómo lo hace.

Abstracción

- En un LPOO, la funcionalidad de una estructura de datos se expresa a través de la interfaz pública de la clase o clases asociadas, que definen la estructura de datos.
- Interfaz pública, se refiere a las definiciones (nombres, tipos de argumentos y tipos de retorno) de los métodos de una clase.
- Esta es la única parte de la clase a la que se puede acceder por un usuario.

Encapsulación

- El concepto de **encapsulación**, establece que los diferentes componentes de un sistema de software no deben revelar los detalles internos de sus respectivas implementaciones.
- Una de las principales ventajas de la encapsulación es que da libertad al programador de los detalles de un sistema en la implementación.
- La única restricción para el programador es mantener la interfaz abstracta.

Modularidad

- Los sistemas de software consisten de varios componentes que deben interactuar correctamente para que todo el sistema funcione apropiadamente.

- La **modularidad** se basa en el principio de estructuración de código, que consiste en que los diferentes componentes de un sistema de software se dividen en unidades funcionales.

Unified Modeling Language (UML)

- Formalismo estándar de facto para el análisis y diseño de software.
- **Lenguaje de modelado,** permite la **representación conceptual** y física de un sistema.
- Estándar por ***Object Management Group*** *(OMG).*
- Utiliza notaciones gráficas para expresar el Análisis y Diseño Orientado a Objeto de proyectos de software.
- Simplifica el complejo proceso de diseño de software.

UML

- Herramientas CASE para modelar en UML:
  - Rational Rose
  - Together
  - Poseidón
  - ArgoUML
  - **StarUML**
  - Architect Enterprise

El modelo Orientado a Objetos

- Los modelos de datos son la herramienta principal para ofrecer la abstracción.
- Un modelo basado en la percepción de una colección de objetos.

El modelo Orientado a Objetos

- Un programa (independientemente del lenguaje en el cuál este escrito) está constituido por dos partes fundamentales:
  - **Objetos**: Una representación de la información (datos) relativos al dominio de interés.
  - **Operaciones**: Una descripción de cómo manipular la representación, de tal manera que se realice la funcionalidad deseada.

El modelo Orientado a Objetos

- Una **entidad** es un **objeto** que existe y que se puede distinguir de otros objetos, es decir, una "cosa" del mundo real con existencia independiente.

- Una entidad puede ser concreta (como una persona o un libro), o puede ser abstracta como un concepto (un curso universitario o un puesto de trabajo).

El modelo orientado a objetos

- Un **objeto** se caracteriza por tener un **estado** y un **comportamiento**.
  - El **estado** corresponde a los valores que toman un conjunto de **atributos** o variables de instancia
  - El **comportamiento** es llevado a cabo mediante una serie de operaciones que se aplican sobre el objeto, y que se denominan **métodos**.

El modelo orientado a objetos

- Los **objetos** que tienen el mismo tipo de atributos y el mismo comportamiento son **agrupados en clases**, que se organizan en un diagrama o jerarquía de clases, en donde las clases pueden estar relacionadas mediante **relaciones de asociación** o mediante **relaciones de herencia.**

El modelo orientado a objetos

- La única forma en la que un objeto puede acceder a los datos de otro objeto es a través de los **métodos** de este objeto. Esto se denomina **envío de mensajes** al objeto.
- La interfaz de llamada mediante los métodos de un objeto define la **parte visible**, mientras que la **parte interna del objeto** (variables y código de los métodos) **no es visible** externamente. De esta forma se tienen dos niveles de abstracción.

El modelo orientado a objetos

- Sea un objeto que representa una cuenta, y que tal objeto contiene las variables de instancia Num_cuenta y saldo. Este objeto puede tener un método denominado Depositar() que añade una cantidad al saldo, Retirar().

Visibilidad

- Niveles distintos de encapsulación se pueden establecer para los miembros de una clase (atributos y métodos) en función de su acceso:

Visibilidad	Significado	Java	UML
Pública	Se puede acceder al miembro de la clase desde cualquier lugar.	`public`	+
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.	`protected`	#
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	`private`	-

Visibilidad

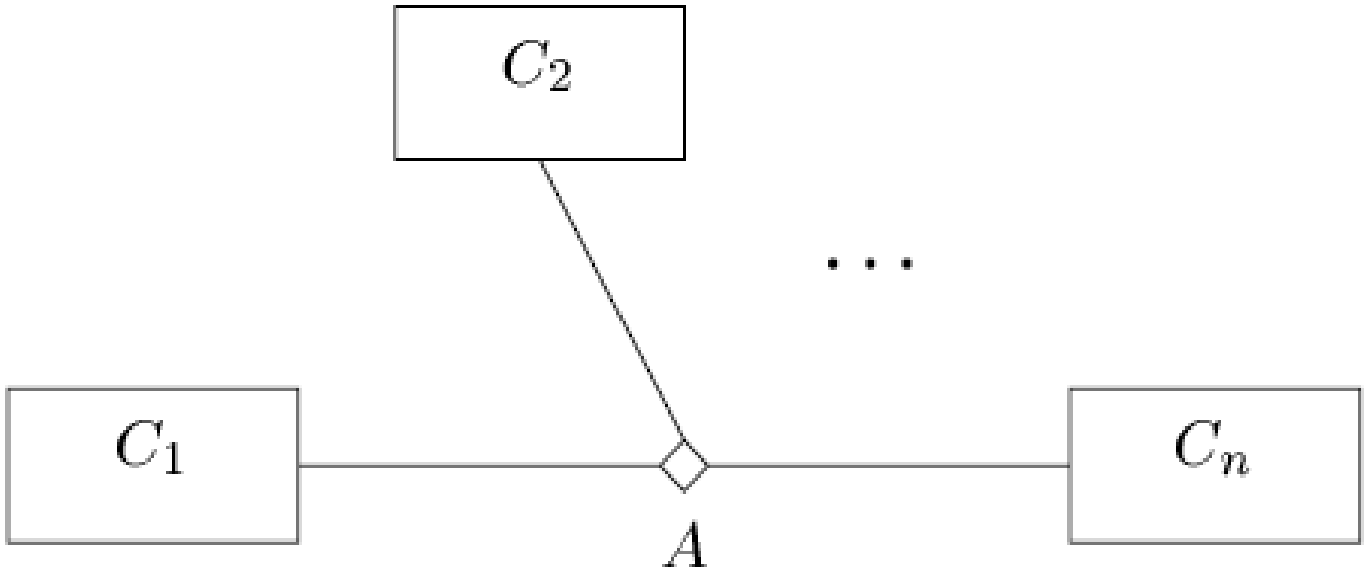
- Para encapsular por completo el estado de un objeto, todos sus atributos se declaran como variables de instancia privadas.
- A un objeto siempre se le accede a través de sus métodos públicos (su **interfaz**).
- Para utilizar un objeto no es necesario saber qué algoritmos utilizan sus métodos ni qué tipos de datos se emplean para mantener su estado (su **implementación**).

Relación de Asociación

- Las relaciones de asociación suelen ser bidireccionales (se pueden recorrer en ambos sentidos), en ocasiones es deseable hacerlas unidireccionales (restringir su navegación en un único sentido).

Relación de asociación

- Las **asociaciones** suelen ser **bidireccionales**, también pueden relacionarse con varias clases C_1, C_2,...,C_n es decir, relación de **asociación n-aria.**



Cardinalidad o Multiplicidad

- La **cardinalidad** de una asociación determina cuántos objetos de cada tipo intervienen en la relación.
  - El número de instancias de una clase que se relacionan con una instancia de la otra clase.
  - Cada asociación tiene dos multiplicidades (una para cada extremo de la relación).
  - Para especificar la multiplicidad de una asociación hay que indicar la multiplicidad mínima y la máxima (mínima..máxima)

Cardinalidad o Multiplicidad

Multiplicidad	Significado
`1`	Uno y sólo uno
`0..1`	Cero o uno
`N..M`	Desde N hasta M
`*`	Cero o varios
`0..*`	Cero o varios
`1..*`	Uno o varios (al menos uno)

Cardinalidad o Multiplicidad

- Cuando la cardinalidad es mínima 0, la relación es opcional.
- Una cardinalidad mínima mayor o igual que 1 establece una relación obligatoria.

Todo profesor pertenece a una Facultad.

A una Facultad pueden pertenecer varios profesores.

Cardinalidad o Multiplicidad

Relación obligatoria

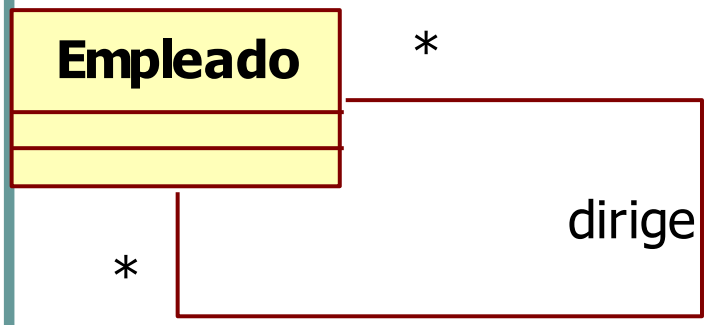
Una cuenta ha de tener un titular como mínimo

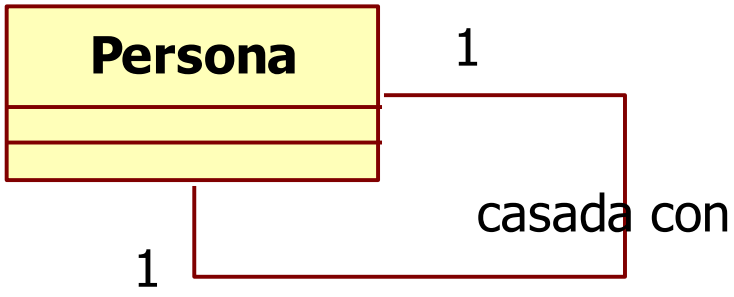
Relación opcional

Un cliente puede o no ser titular de una cuenta

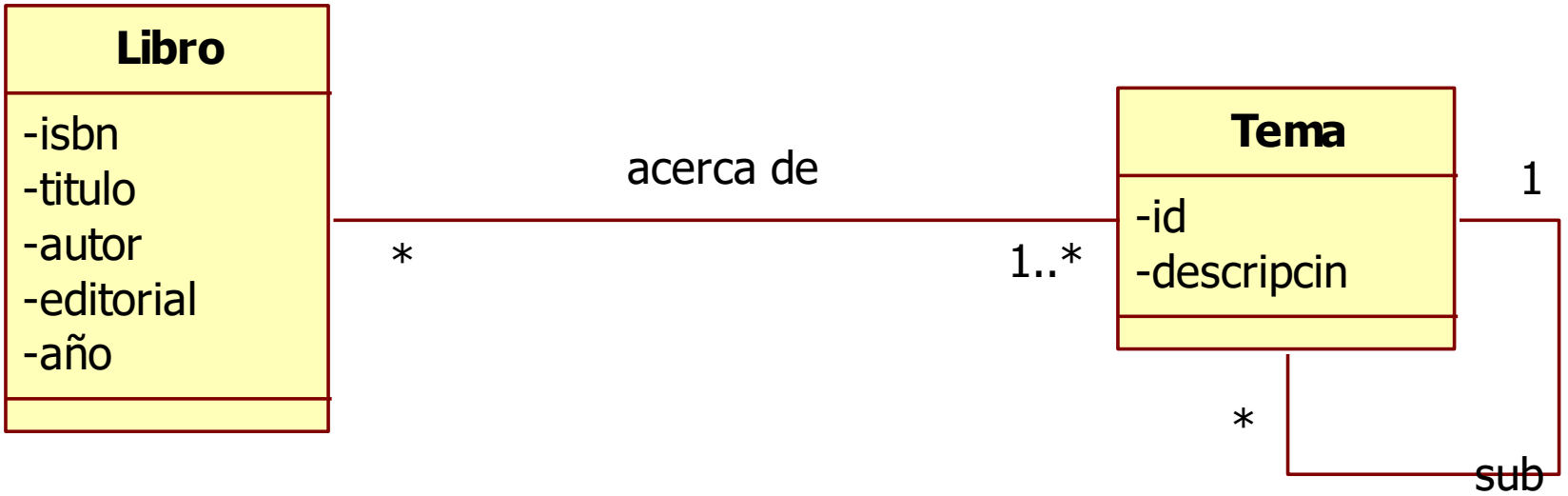
Cardinalidad o Multiplicidad

- **Relaciones involutivas**
  - Cuando la misma clase aparece en los dos extremos de la asociación.

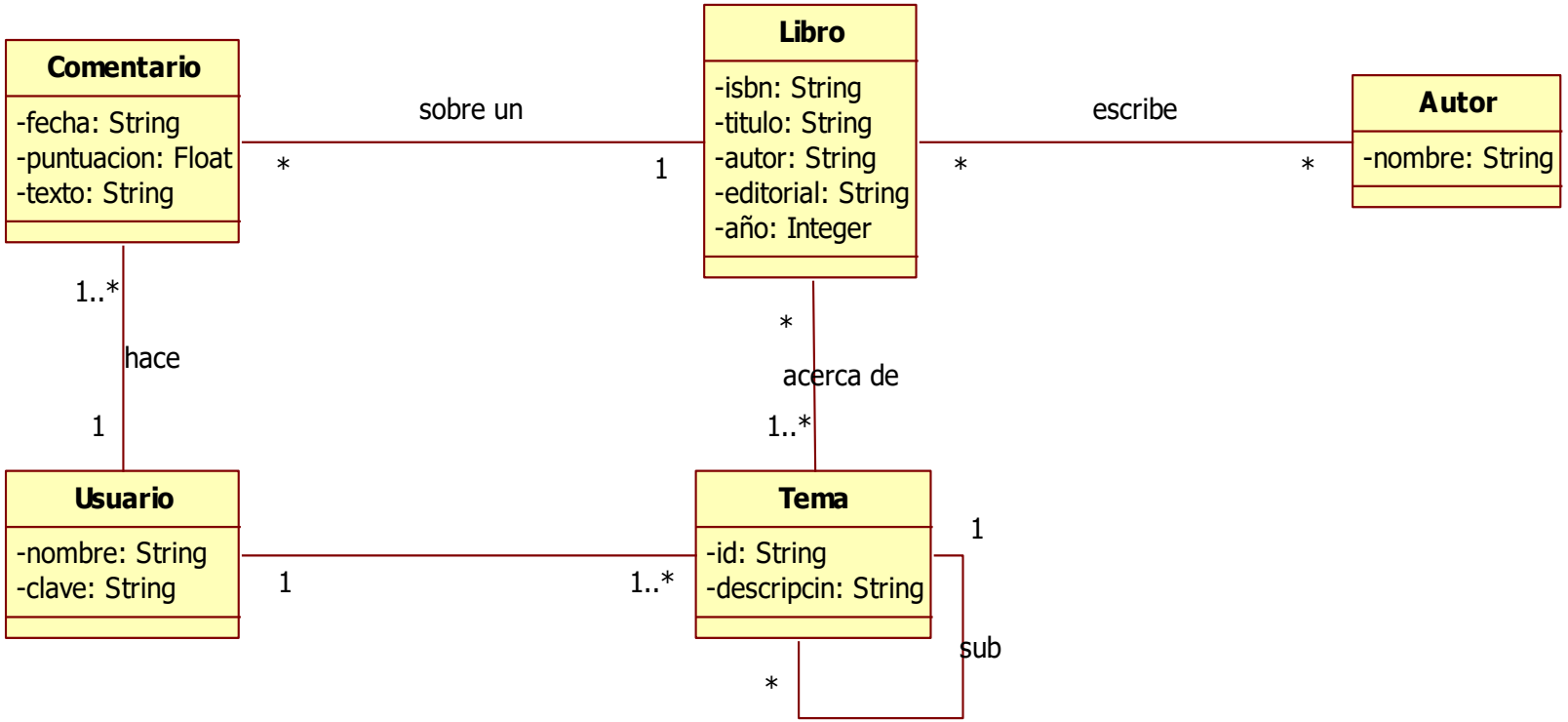




Ejemplo


Libro
-isbn
-titulo
-autor
-editorial
-año
acerca de
*
1..*
Tema
-id
-descripcin
1
*
sub

Ejemplo

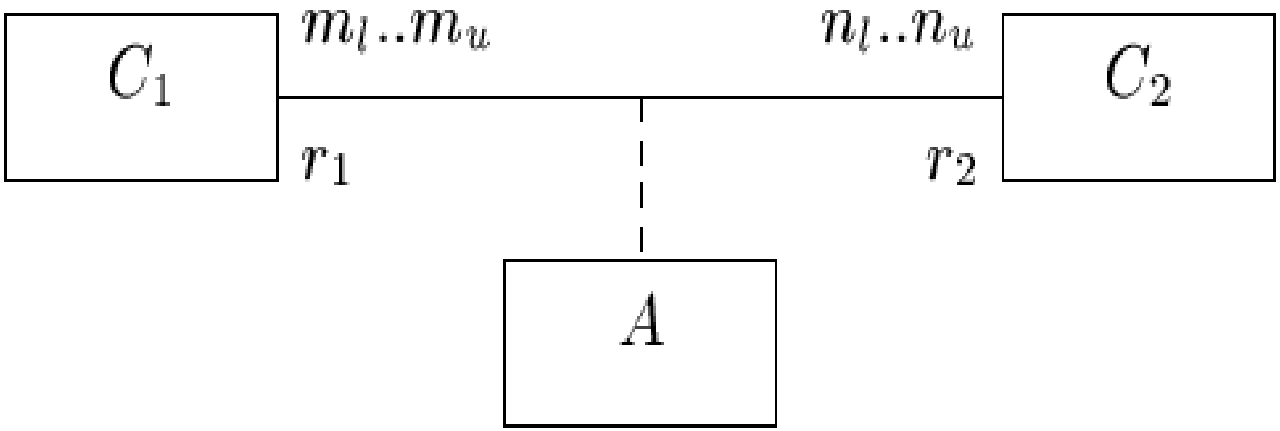

Comentario
-fecha: String
-puntuacion: Float
-texto: String
sobre un
*
1
Libro
-isbn: String
-titulo: String
-autor: String
-editorial: String
-año: Integer
escribe
*
*
Autor
-nombre: String
1..*
hace
1
*
acerca de
1..*
Usuario
-nombre: String
-clave: String
1
1..*
Tema
-id: String
-descripcin: String
1
sub
*

Clase de asociación

- Una relación de asociación que tiene **una clase de asociación describe** las propiedades de **la asociación**, como atributos, operaciones, etc., de una asociación binaria entre dos clases C_1 y C_2, con **una clase de asociación** A.

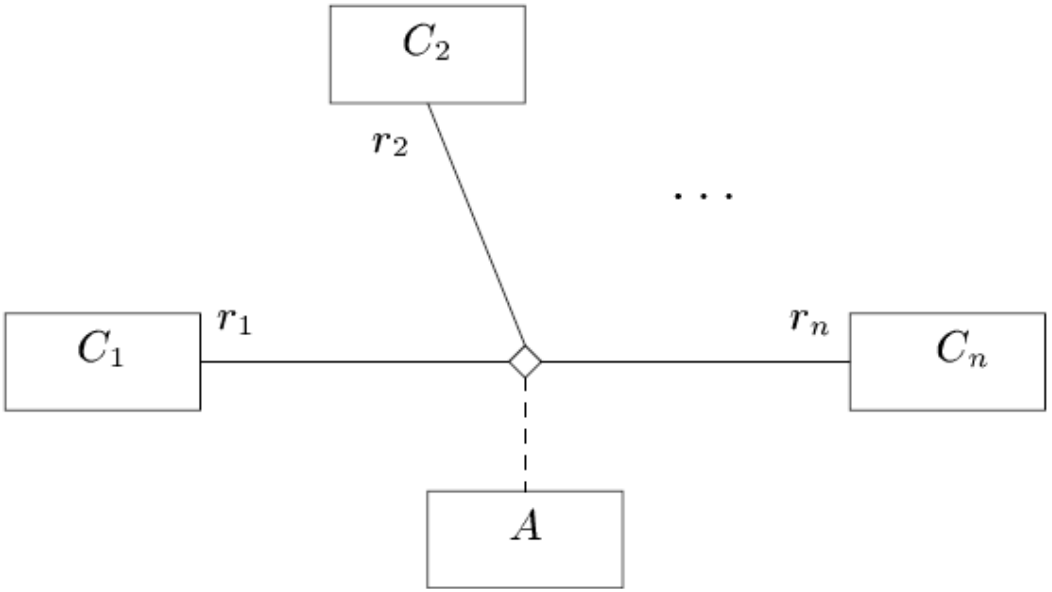
Clase de asociación

- La clase A es la **clase de asociación** relacionada con la asociación, y la r_1 y r_2 son los nombres de función de C_1 y C_2, respectivamente, especifican el papel que juega cada clase dentro de la Asociación A.

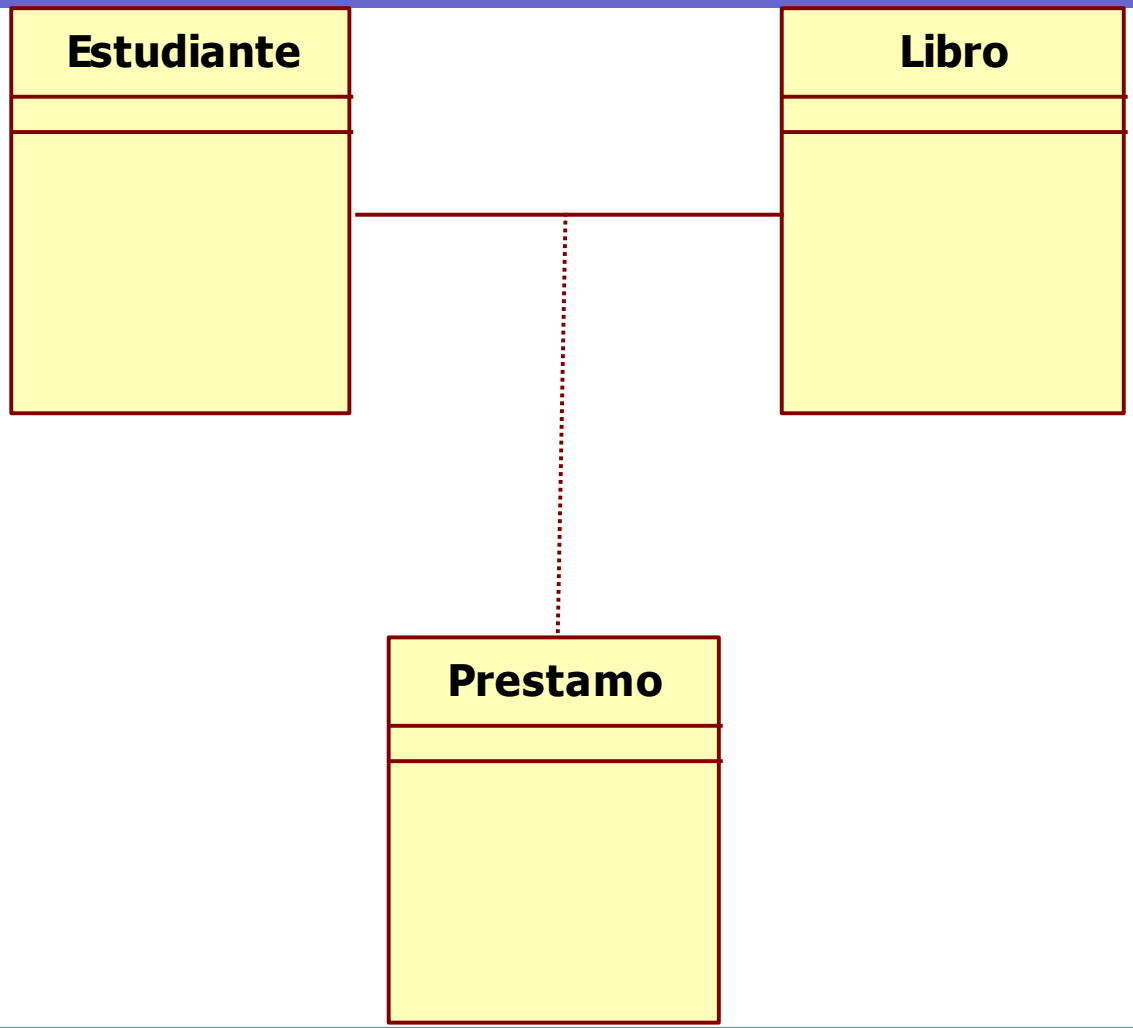


Clase de asociación

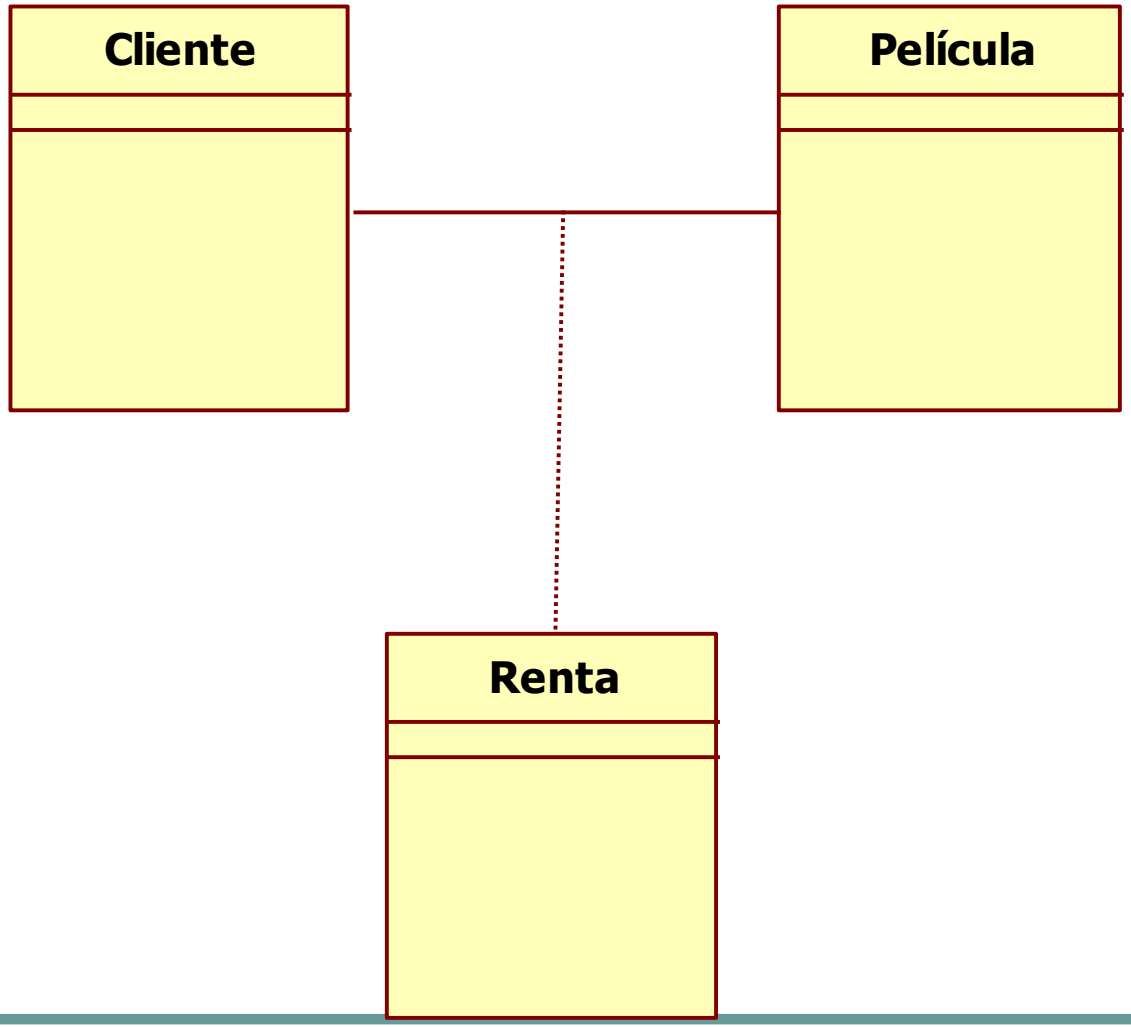
- Una clase de asociación también se puede agregar a una asociación n-aria.



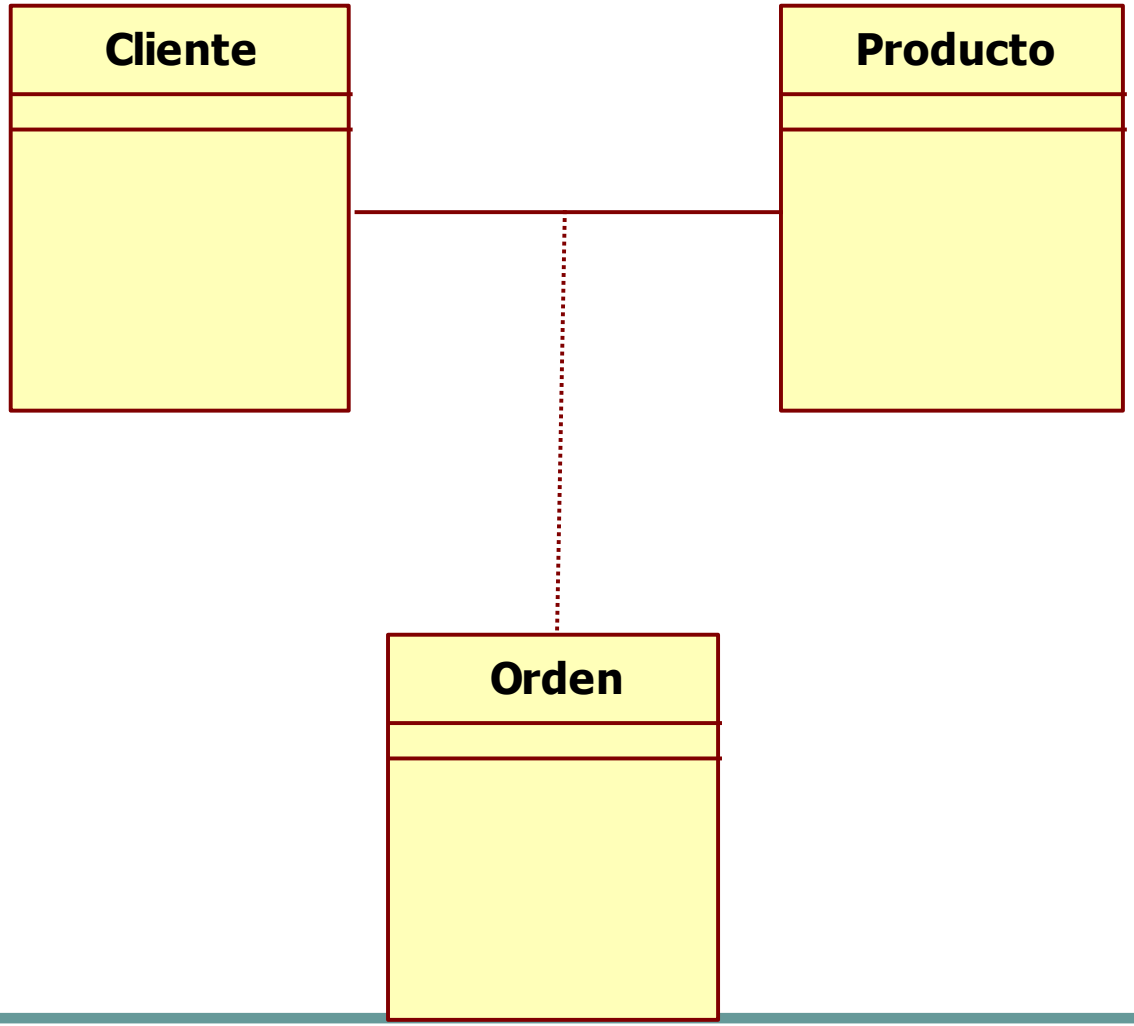
Ejemplo



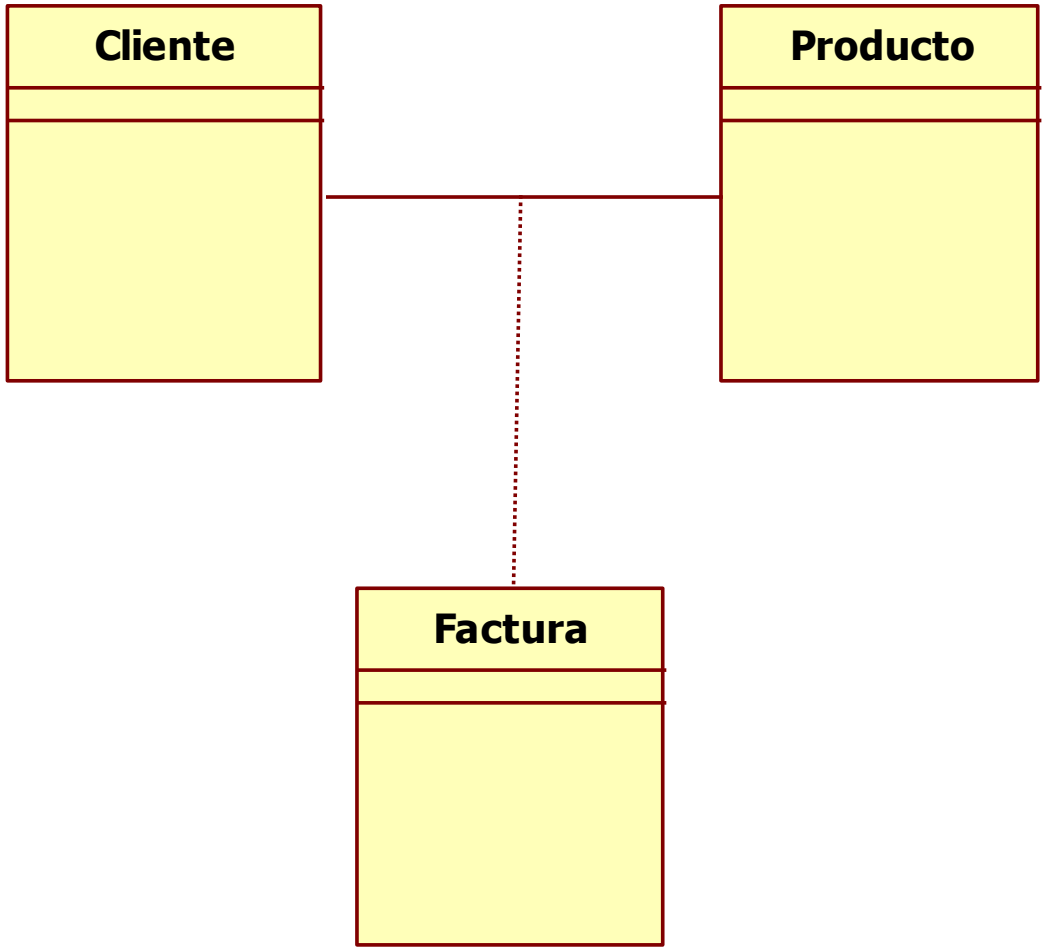
Ejemplo



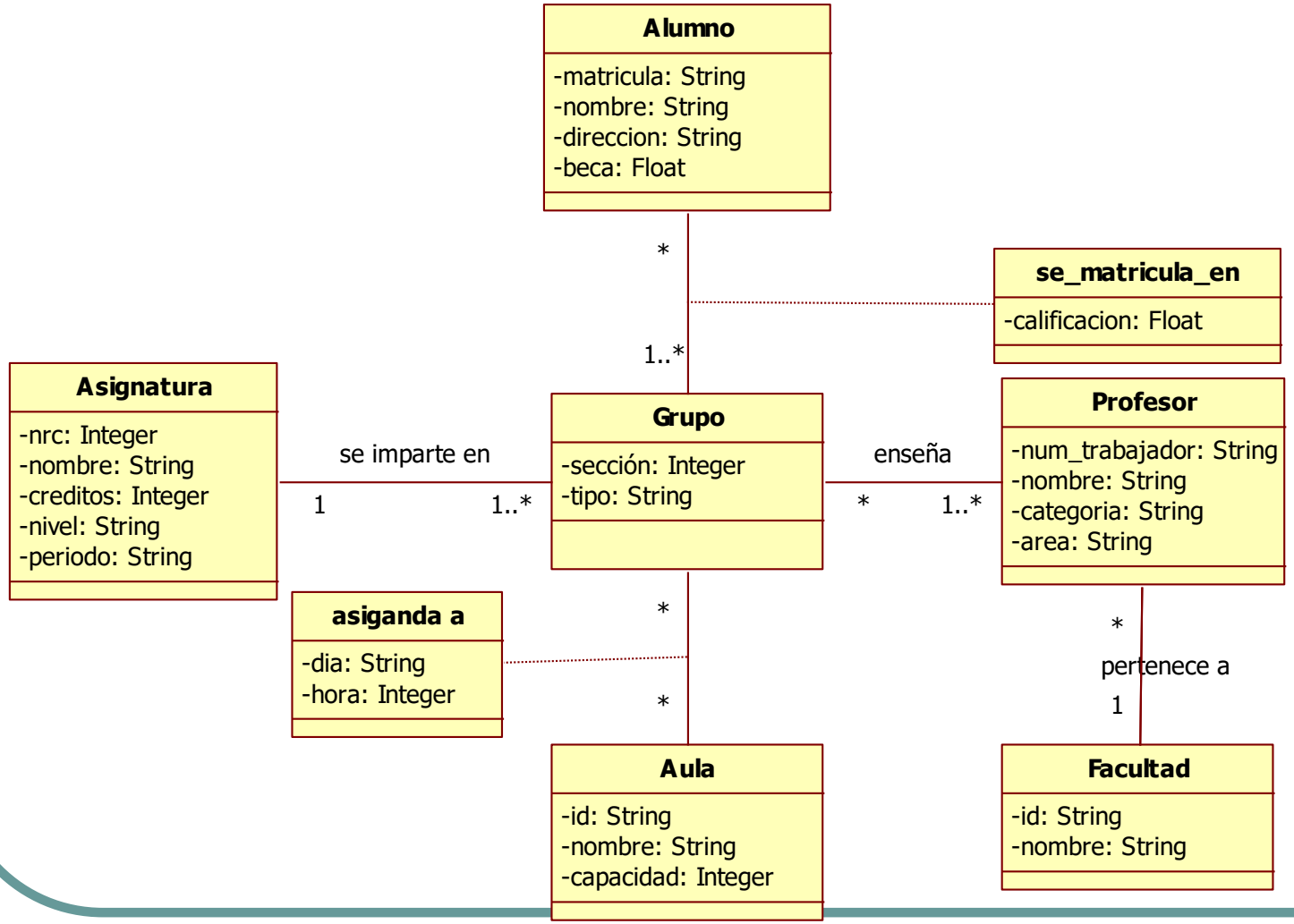
Ejemplo



Ejemplo



Ejemplo


Alumno
-matricula: String
-nombre: String
-direccion: String
-beca: Float
*
1..*
se_matricula_en
-calificacion: Float
Asignatura
-nrc: Integer
-nombre: String
-creditos: Integer
-nivel: String
-periodo: String
se imparte en
1
1..*
Grupo
-sección: Integer
-tipo: String
enseña
*
1..*
Profesor
-num_trabajador: String
-nombre: String
-categoria: String
-area: String
asiganda a
-dia: String
-hora: Integer
*
*
*
pertenece a
1
Aula
-id: String
-nombre: String
-capacidad: Integer
Facultad
-id: String
-nombre: String

Herencia

- La **herencia** permite la definición de clases a partir de clases existentes, heredando a las nuevas clases los atributos y el comportamiento de las clases existentes, cumpliéndose también que todo objeto de una subclase también es objeto de su superclase.

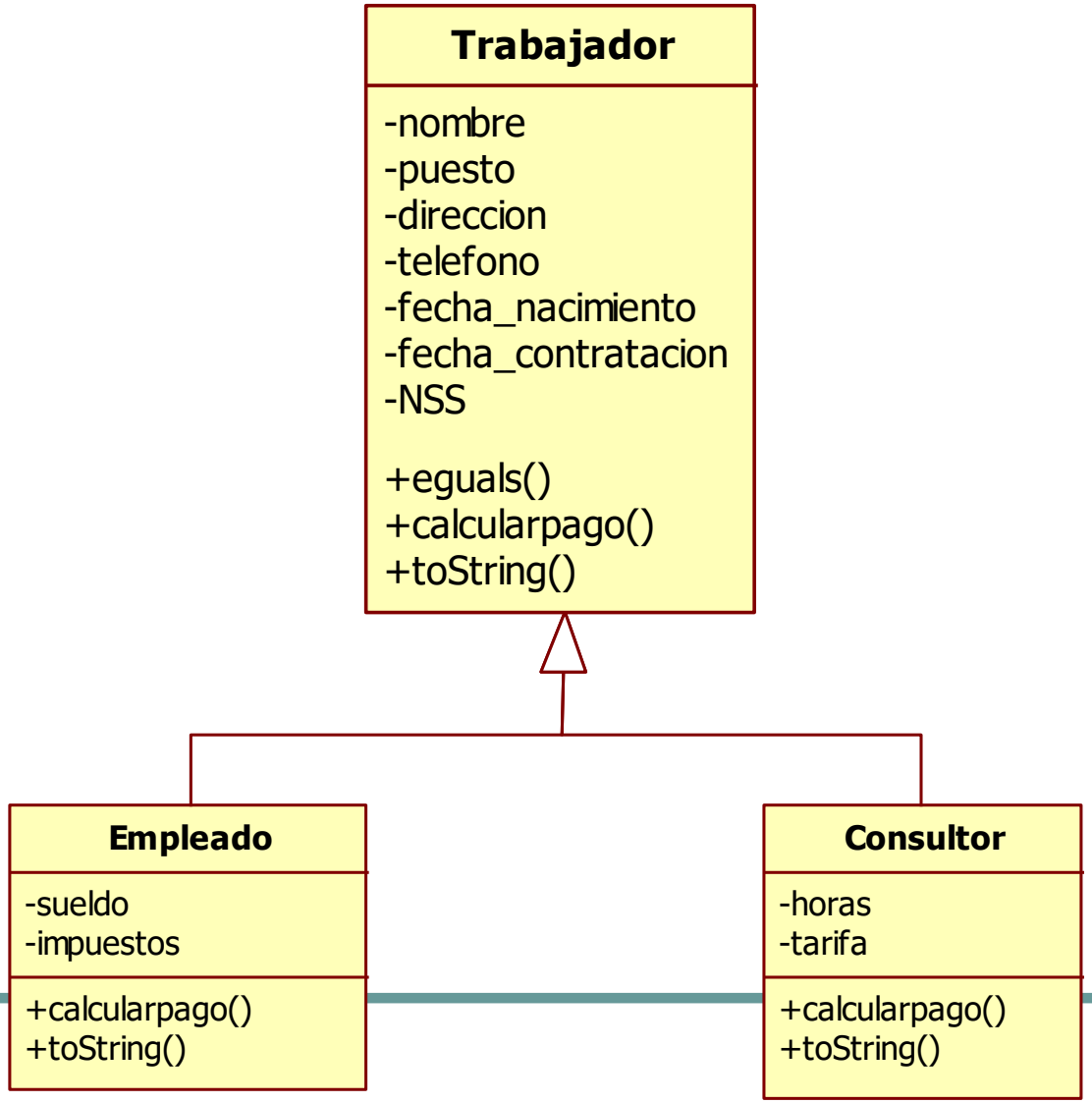
Herencia

- Los objetos de la nueva clase **heredan** los **atributos** y los **métodos** de la superclase.
- La **subclase,** puede poseer atributos y métodos que no existen en la clase original.
- La **herencia** permite reutilizar clases, ya que se crea una nueva clase que extiende la funcionalidad de una clase existente sin tener que reescribir el código asociado a esta última.

Herencia

- **Trabajador**, clase genérica que almacena datos como nombre, dirección, número de teléfono o el número de seguro social de un trabajador.
- **Empleado**, clase que representa a los empleados de nómina mensual (encapsula datos como su salario o las retenciones).
- **Consultor,** clase que representa a los trabajadores que cobran por hora (registra el número de horas de trabajo y la tarifa).

Ejemplo


Trabajador
-nombre
-puesto
-direccion
-telefono
-fecha_nacimiento
-fecha_contratacion
-NSS
+eguals()
+calcularpago()
+toString()
Empleado
-sueldo
-impuestos
+calcularpago()
+toString()
Consultor
-horas
-tarifa
+calcularpago()
+toString()

Polimorfismo

- Posibilidad de definir clases diferentes, que tienen métodos o atributos denominados de forma idéntica, pero que se comportan de manera distinta.

Agregación y Composición

- Son casos particulares de asociaciones: relación **entre un todo y sus partes**.

Relación de Agregación

- Las partes pueden ser de distintos agregados.
- Una **agregación** es una relación binaria entre las instancias de dos clases, lo que denota una relación **parte-todo**, es decir, una relación que especifica cada instancia de una clase que contiene un conjunto de instancias de otra clase.

Relación de Agregación

- Es común que un objeto contenga objetos de otras clases, tal relación se conoce como **Agregación**.
  - Un objeto **Auto *tiene*** o ***esta*** conformado por objetos de otras clases Llantas, Motor, etc.
- La relación de **Agregación** se puede leer como **tiene un**, **es parte de** o **consta de.**
  - Un auto tiene un **motor**

Relación de Agregación

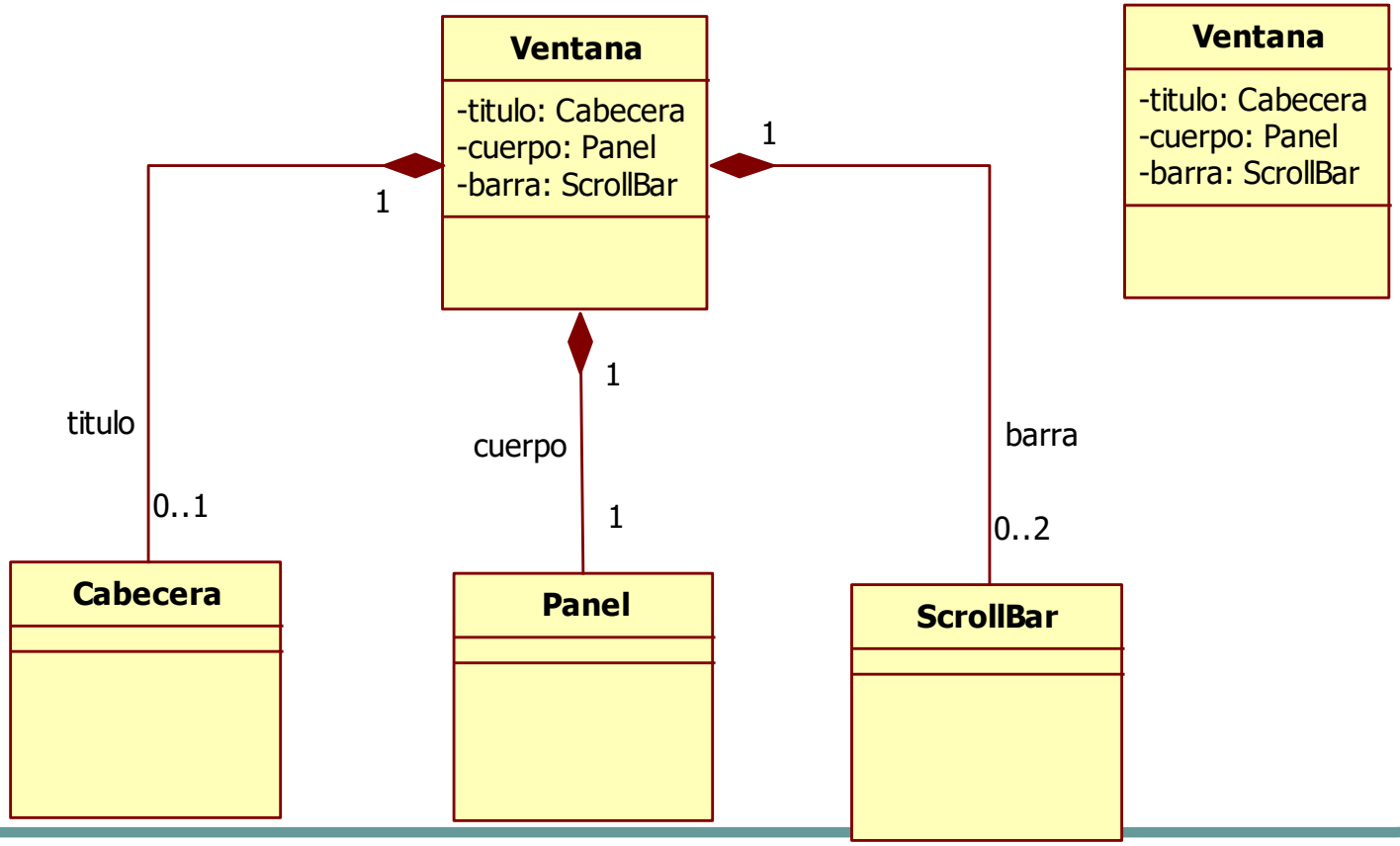
- La relación de **agregación**, es una relación **débil** entre los objetos, ya que estos pueden existir independientemente del **todo**.
- Es decir, un motor **puede existir**, sin ser parte de un auto.

Relación de Composición

- Agregación disjunta y estricta: Las partes sólo existen asociadas al compuesto (sólo se accede a ellas a través del compuesto).
- La relación de composición es muy parecida a la de Agregación (del tipo **tiene un**), sólo que en este caso la relación es mas fuerte.

Relación de Composición

- La **relación de Composición** se denota por una línea terminada en un **rombo coloreado.**



Relación de Dependencia

- Es una relación semántica entre dos clases (conjuntos de objetos), en la cual un cambio en una clase puede afectar a la semántica de la otra.
- Relación entre un cliente y el proveedor de un servicio usado por el cliente.
  - Cliente es el objeto que solicita un servicio.
  - Proveedor es el objeto que provee el servicio solicitado.

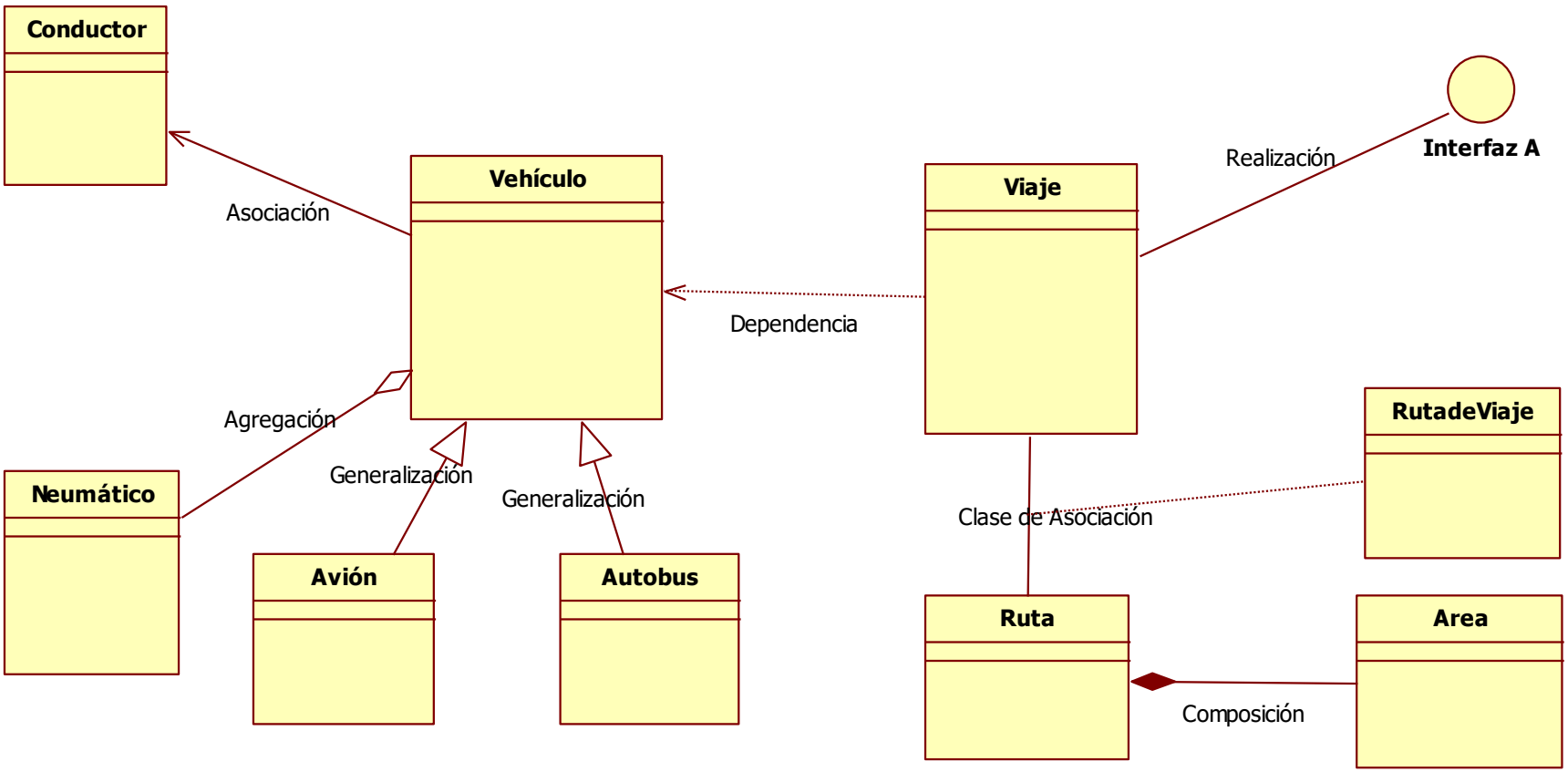
Relación de Dependencia

- Para resolver una ecuación de segundo grado se recurre a sqrt de la clase Math para calcular una raíz cuadrada.

Representación del dominio: Objetos

1. **Identificar los objetos** que intervienen en el dominio.
2. Establecer los **atributos de los objetos**.
3. **Agrupar los objetos** del mismo tipo en **clases**.
4. Establecer las **relaciones entre las clases**; es decir, cómo los objetos de las diferentes clases están conectados.
5. Definir las clases, las propiedades de las clases y las relaciones entre las clases en el lenguaje de programación seleccionado.

Ejemplo


Conductor
Asociación
Vehículo
Dependencia
Viaje
Realización
Interfaz A
Agregación
Neumático
Generalización
Avión
Generalización
Autobus
Clase de Asociación
RutadeViaje
Ruta
Composición
Area