11ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 2

Principios y Patrones de Diseño

Mtro. ISC Miguel Ángel Sánchez Hernández

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

50 AÑOS Educación Continua 1971 - 2021

Convenios

- Tolerancia de inicio de clases 15 minutos
- 20 minutos de receso
  - Viernes 18:30-18:50
  - Sábados 11:30-11:20

Evaluación

• Tareas	30%
• Prácticas en clase	30%
• Avance de Proyecto(UML y Código)	40%
	100%

* Tu **calificación final** será valida si entregas el avance del proyecto, este debe incluir UML (StarUML)y un avance de código(proyecto de eclipse) aplicando los principios de SOLID y Patrones de Diseño si fuera el caso.

Objetivo

Entender los principios de la Programación Orientada a Objetos para comprender los Patrones de Diseño

Lo que veremos

- Principios de Diseño Orientado a Objetos
- Terminología POO
- Relaciones entre clases
- Ocupar UML para las relaciones entre clases

Terminología de POO

Una clase es una plantilla o plano a partir del cual se crean realmente los objetos. Una clase tiene atributos(campos de clase) y métodos.

Las expresiones de creación de instancias(ejemplares) de clases que empiezan con la palabra clave **new** crean nuevos objetos.

Cuando cada objeto de una clase mantiene su propia copia de un atributo, el campo que representa a ese atributo también se conoce como **variable de instancia**. Cada objeto (instancia) de la clase tiene una instancia separada de la variable en la memoria.

Al proceso de declarar variables de instancia con el modificador de acceso private se le conoce como **ocultamiento de datos o encapsulación**.

Terminología de POO

La clave para que funciones la encapsulación consiste en hacer que los métodos nunca accedan directamente a los campos de clases distintas de la propia.

Una ventaja de los campos es que todos los métodos de la clase pueden usarlos. Otra diferencia entre un **campo** y una **variable local** es que un campo tiene un valor inicial predeterminado, que Java proporciona cuando el programador no especifica el valor inicial del campo, pero una variable local no hace esto.

Un objeto especifico que sea una instancia(ejemplar) de una clase, tendrá valores específicos para sus atributos, a este conjunto de valores del objeto se denomina **estado actual** del objeto.

Terminología de POO

Los programas solo deben interactuar con los datos de otros objetos por medio de los métodos de los objetos, a esto también se le llama **mensajes**.

La encapsulación es la forma de dar al objeto su comportamiento de “caja negra”, que es la clave de la **reutilización** y de la **fiabilidad**. Esto es, que una clase puede cambiar radicalmente de forma en que procesa sus datos, pero mientras sigamos ocupando los mismos métodos para su manipulación, ningún otro objeto que se comunique con el saldrá perjudicado.

Trabajar con objetos

Para POO se debe entender tres características importante de los objetos:

- El comportamiento del objeto: ¿Qué se puede hacer con este objeto, o que métodos se le pueden aplicar?.
- El estado del objeto: ¿Cómo reacciona el objeto cuando se le aplican esos métodos?.
- Identidad del objeto: ¿Cómo identificar el objeto que tenga el mismo comportamiento y estos?.

¿Por donde empezar a diseñar en POO?

Si estuviéramos en la programación tradicional, empiezo en un main, pero en POO nosotros buscamos primero las clases y después añadimos los métodos de la clase y sus atributos.

Los sustantivos nos proporcionan una ayuda para encontrar las clases en el análisis del problema y los métodos corresponden a los verbos.

En UML, cada clase se modela en un diagrama de clases en forma de rectángulo con tres compartimientos. El compartimiento superior contiene el nombre de la clase, centrado horizontalmente y en negrita. El compartimiento intermedio contiene los atributos de la clase, que corresponden a los campos en Java. El compartimiento inferior contiene las operaciones de la clase, que corresponden a los métodos y constructores en Java.

¿Ejemplo de un diseño?

Sistema de procesamiento de pedidos; podemos mencionar los siguientes sustantivos que formarían las clases.

- Articulo
- Pedido
- Dirección
- Pago
- Cuenta

Relaciones entre clases

Las relaciones entre clases mas comunes son:

- Dependencia (“utiliza un”)
- Agregación (“tiene un”)
- Herencia (“Es un”)

La relación de dependencia “utiliza un”, es una de las mas evidentes por ejemplo la clase Pedido utiliza la clase Cuenta, porque La clase Pedido necesita acceder al clase Cuente para conocer el crédito. Pero la clase Articulo no necesita la clase Cuenta.

Notación UML de dependencia

¿Nota sobre Relaciones entre clases?

Siempre hay que minimizar el número de clases que tienen relaciones entre si, porque si la clase A no tiene la idea que existe la clase B, y si cambio la clase B, no sabrá nunca que la clase A, que cambio la clase B, además si tiene errores la clase B, no se propagan los errores a la clase A.

Ensamblados

Los ensamblados en particular la **agregación** y la **composición** son formas especiales de asociación entre un todo y sus partes.

La agregación, sus partes en el ensamblado pueden aparecer múltiples veces en el mismo.

En la composición las partes del ensamblado no pueden ser compartidas entre ensamblados.

Ejemplo de Ensamblado

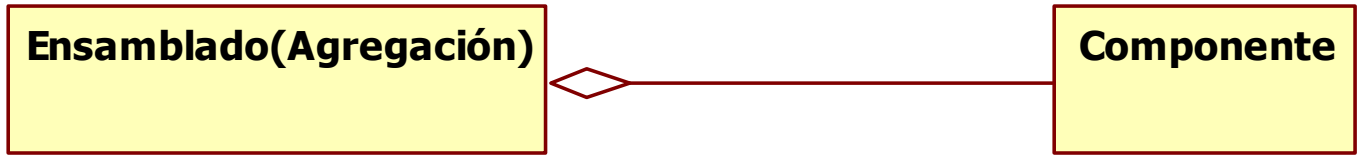
Una red de computadoras se puede considerar un ensamblado, donde las computadoras son sus componentes. Este es un ejemplo de agregación.

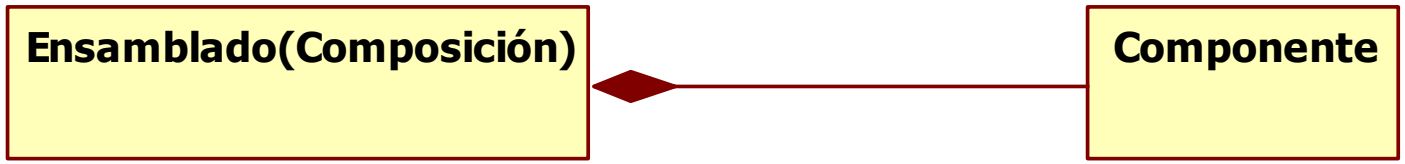
Un automóvil esta constituido por motor, carrocería, llantas etc. Esta es una composición, ya que un motor del automóvil no puede ser compartido por otro.

Propiedades del Ensamblado

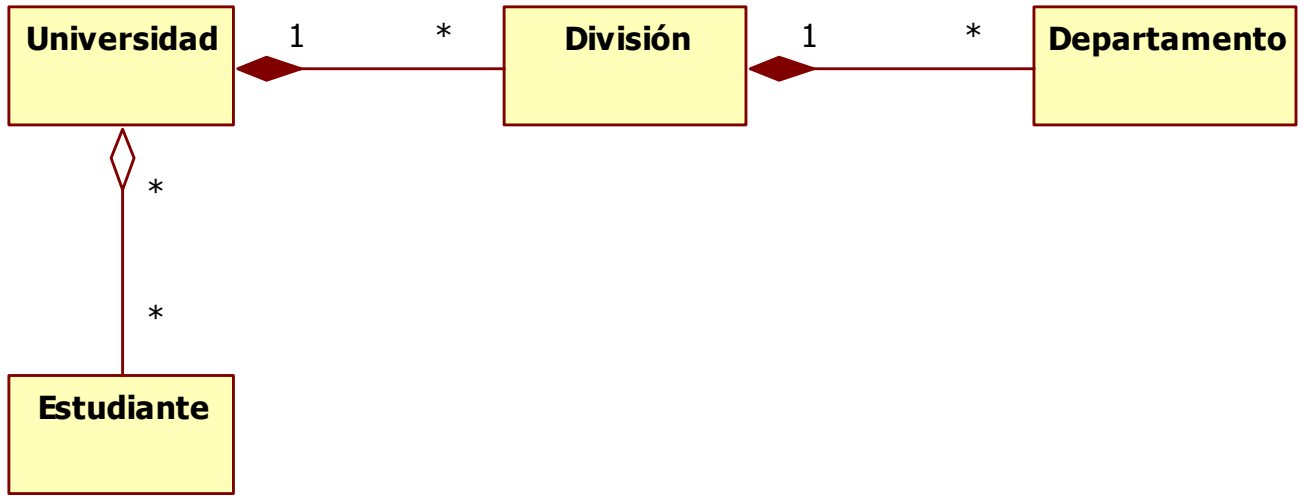
- El ensamblado tiene propiedades de transición. Si A es parte de B y B es parte de C, entonces A es parte de C.
- Es anti simétrico. Si A es parte de B, entonces B no es parte de A.
- Se puede usar la frase “parte-de” o “consiste-de” o “tiene-un”, además algunas operaciones o atributos se pueden propagar a sus componentes.

Notación UML





Ejemplo de Composición y Agregación



Ejemplo de ensamblado de composición con agregación

Herencia

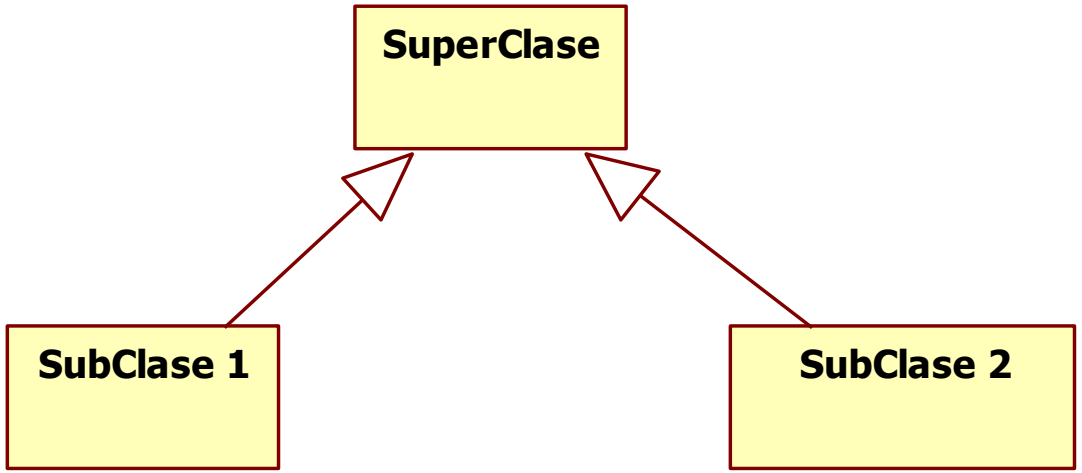
Es una abstracción importante para compartir similitudes entre clases, donde los atributos y operaciones comunes a varias clases se pueden compartir por medio de la superclase más general. Las clases más refinadas se conocen como subclases.

La herencia es una relación “es-una” por ejemplo Impresora láser es una Impresora.

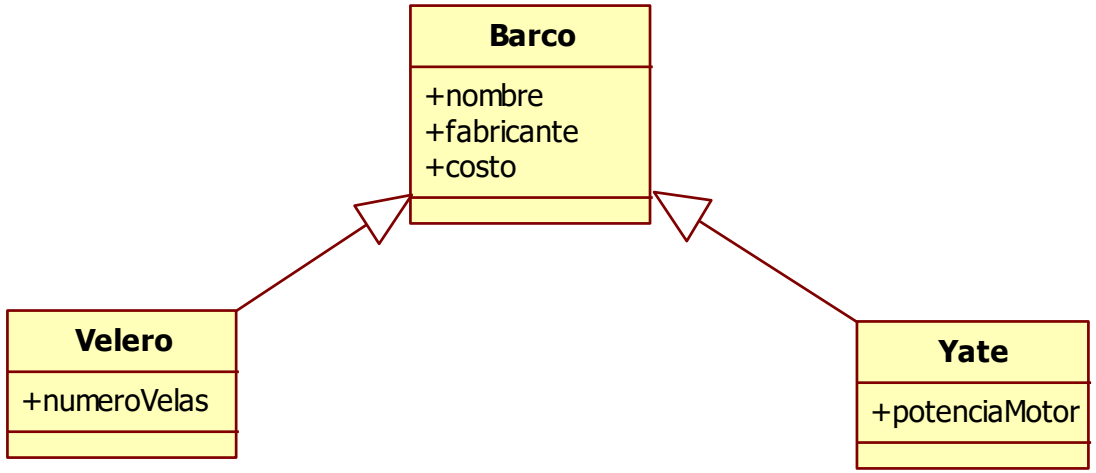
Características de la jerarquía de Herencia

- Los valores de un ejemplar incluyen valores para cada atributo de cada superclase.
- Cualquier operación de cualquier superclase se puede aplicar a un ejemplar.
- Cada subclase no sólo hereda las características de sus superclase, sino también añade sus propios atributos y operaciones.

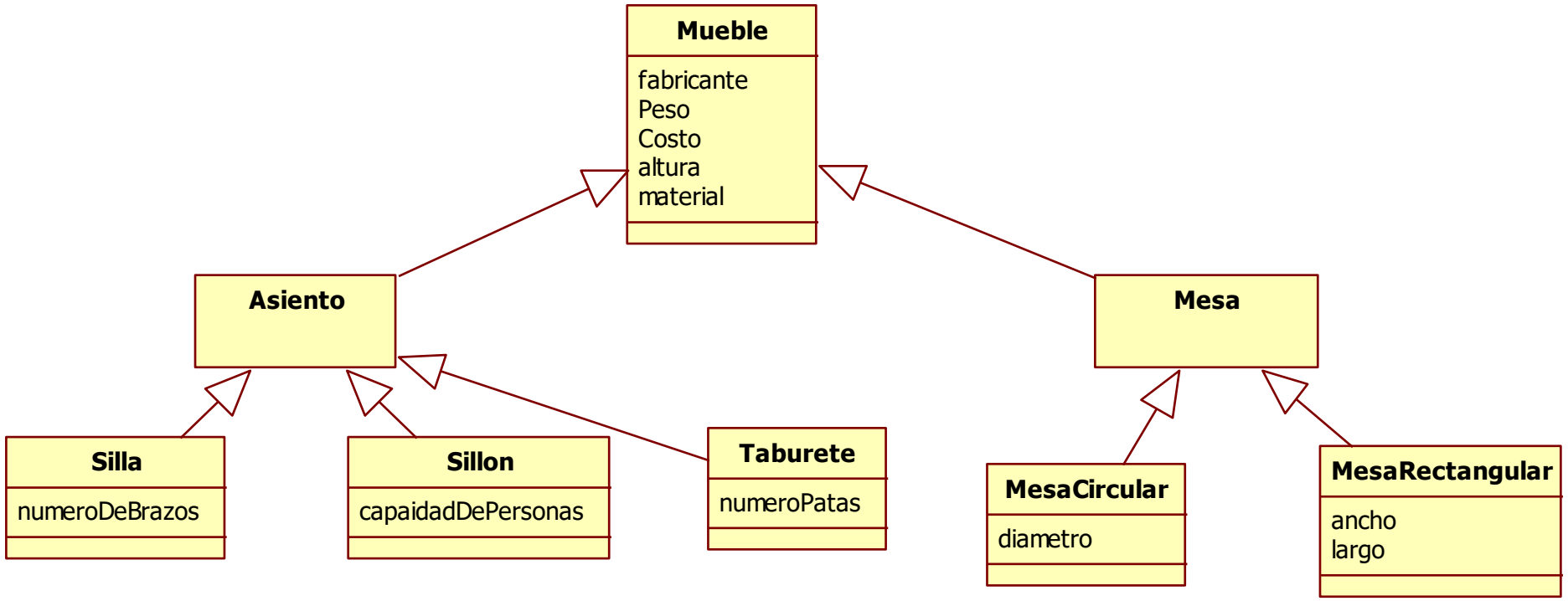
Notación UML de Herencia



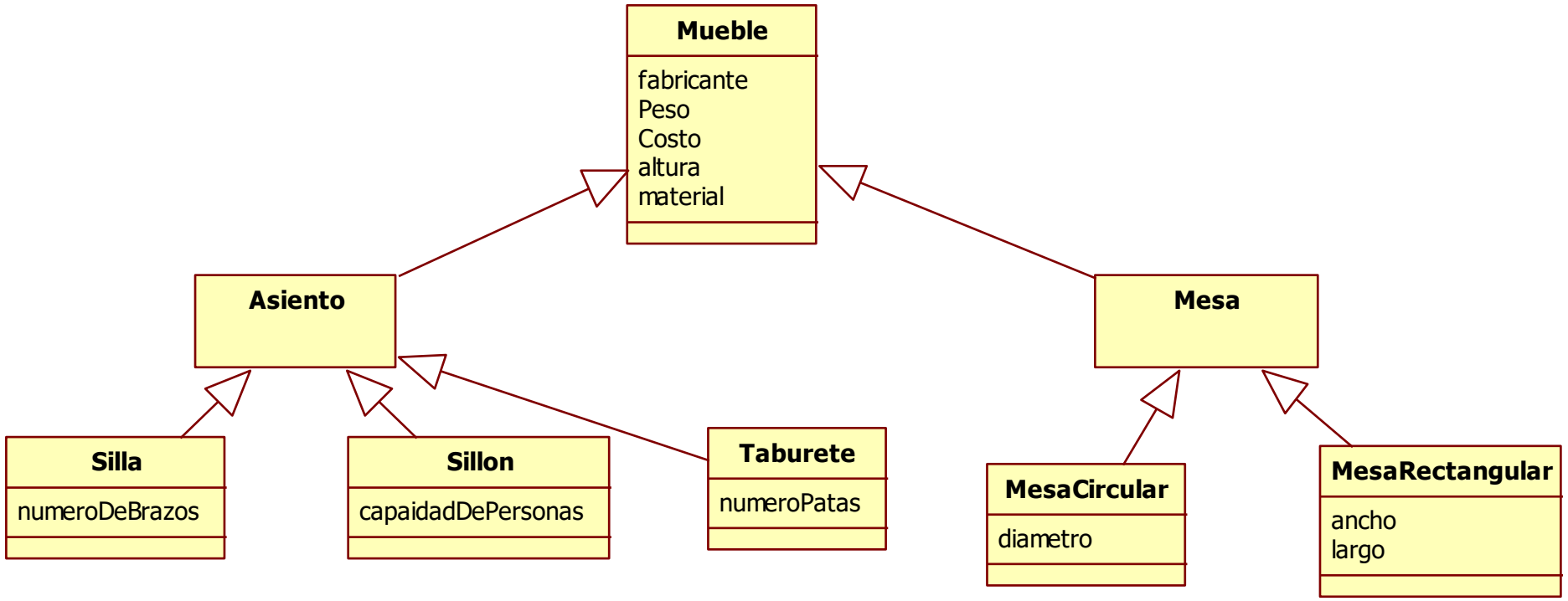
Notación de diagrama de clases de herencia



Ejemplo de Herencia



Ejemplo de Herencia



Nombre a las Asociaciones

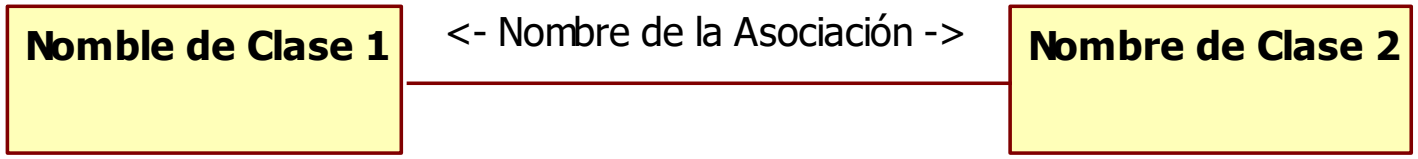
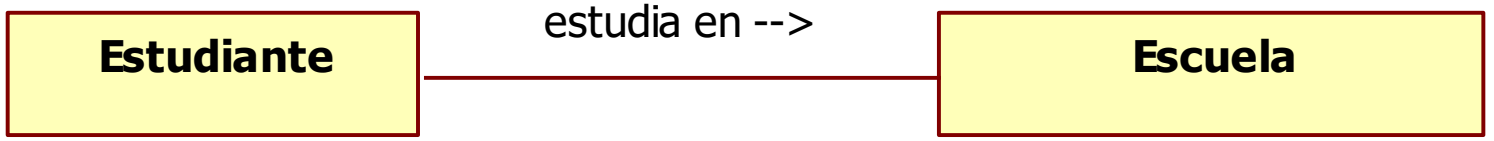
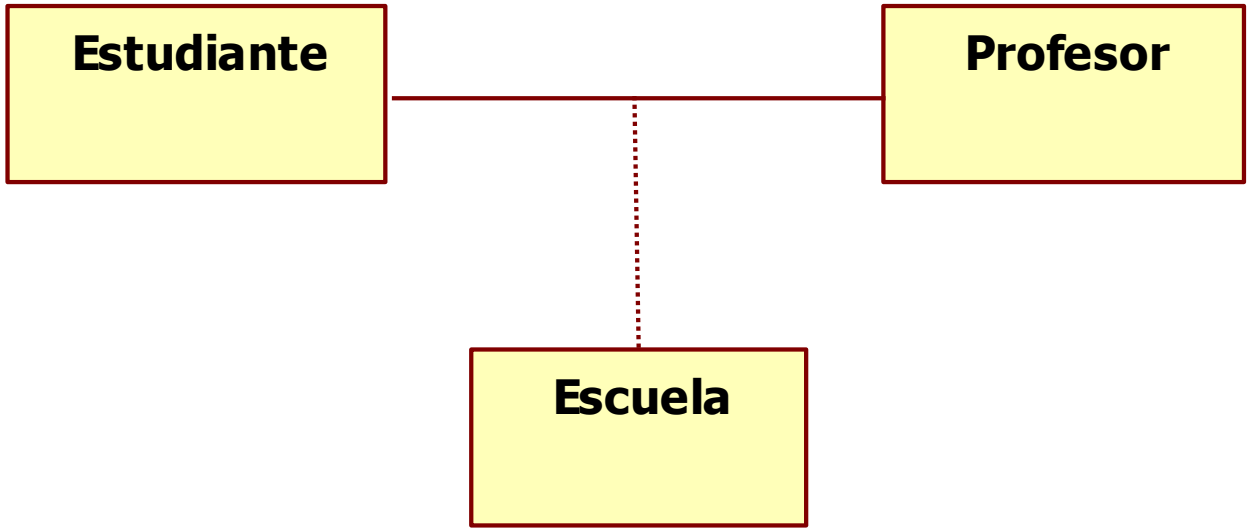


Diagrama de clases que tiene una asociación



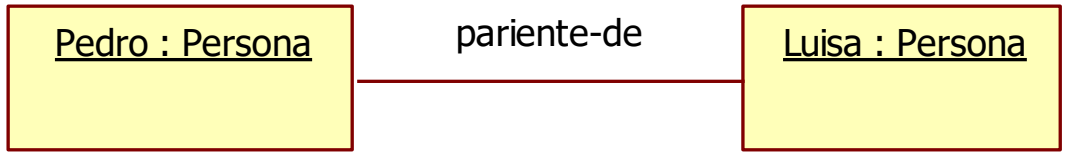
Grado de Asociación

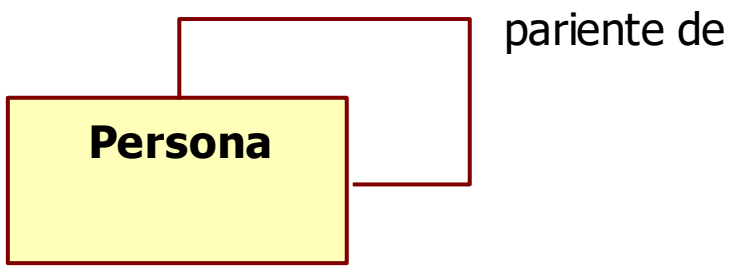
Esta se determina por el número de clases conectadas por la misma asociación, pueden ser binarias, ternarias o grado n.



Asociaciones Reflexivas

Son aquellas que relacionaran objetos de la misma clase.





Multiplicidad

También llamado cardinalidad, con esto especificamos cuántos ejemplares de la clase se pueden relacionar con un ejemplar de otra clase.

Cardinalidad uno-uno

Cardinalidad uno-Muchos

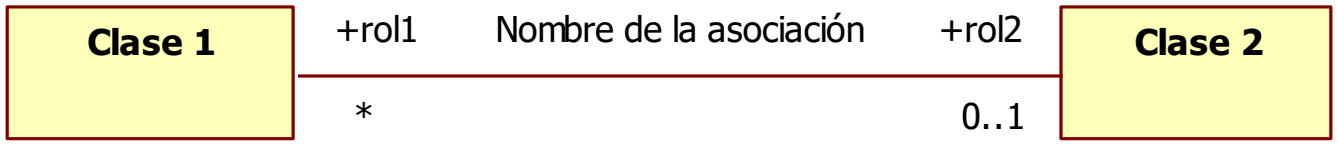
Multiplicidad

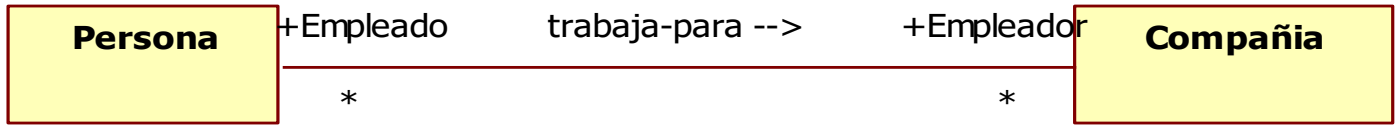
Cardinalidad Muchos-Muchos

Cardinalidad Opcional

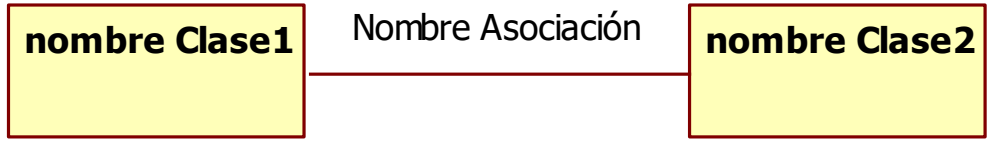
Rol

Un rol describe el papel que juega, cada extremo de una asociación.





Asociaciones y Ensamblados en Java

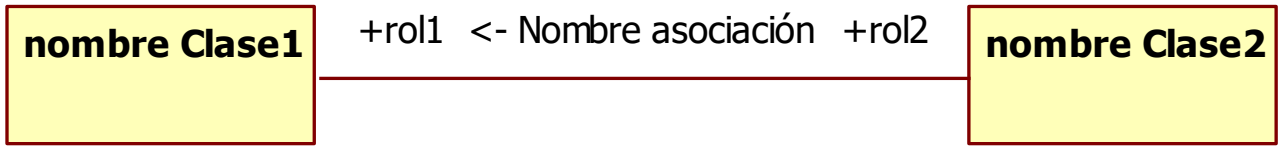



```
class Clase1{
        Clase2 referencia;
}

class Clase2{
        Clase1 referencia;
}
```

Asociaciones y Ensamblados en Java


```
class Clase1{
        Clase2 rol2;
}

class Clase2{
        Clase1 rol1;
}
```

Asociaciones y Ensamblados en Java




```
class Clase1{
}

class Clase2{
        Clase1 rol1;
}
```

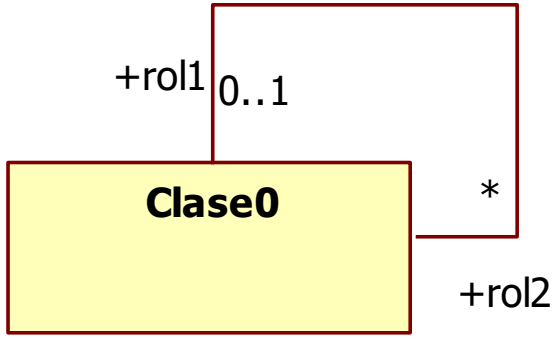
Asociaciones y Ensamblados en Java


```
class Clase1{
        Clase2 rol2[];
}

class Clase2{
        Clase1 rol1;
}
```

Asociaciones y Ensamblados en Java

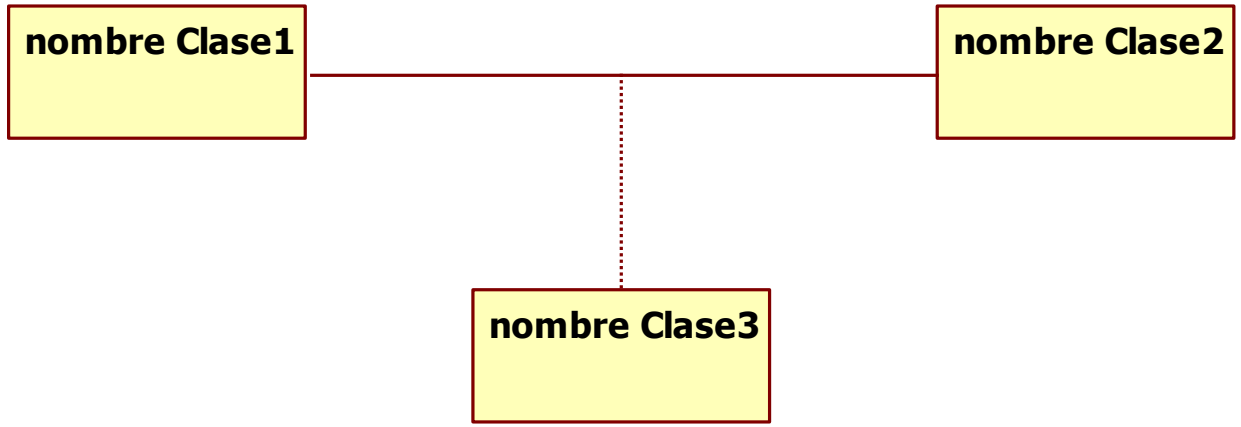

```
class Clase1{
        Clase2 rol2[];
}

class Clase2{
        Clase1 rol1[];
}
```

Asociaciones y Ensamblados en Java




```
class Clase0{
        Clase0 rol1;
        Clase0 rol2[];
}
```

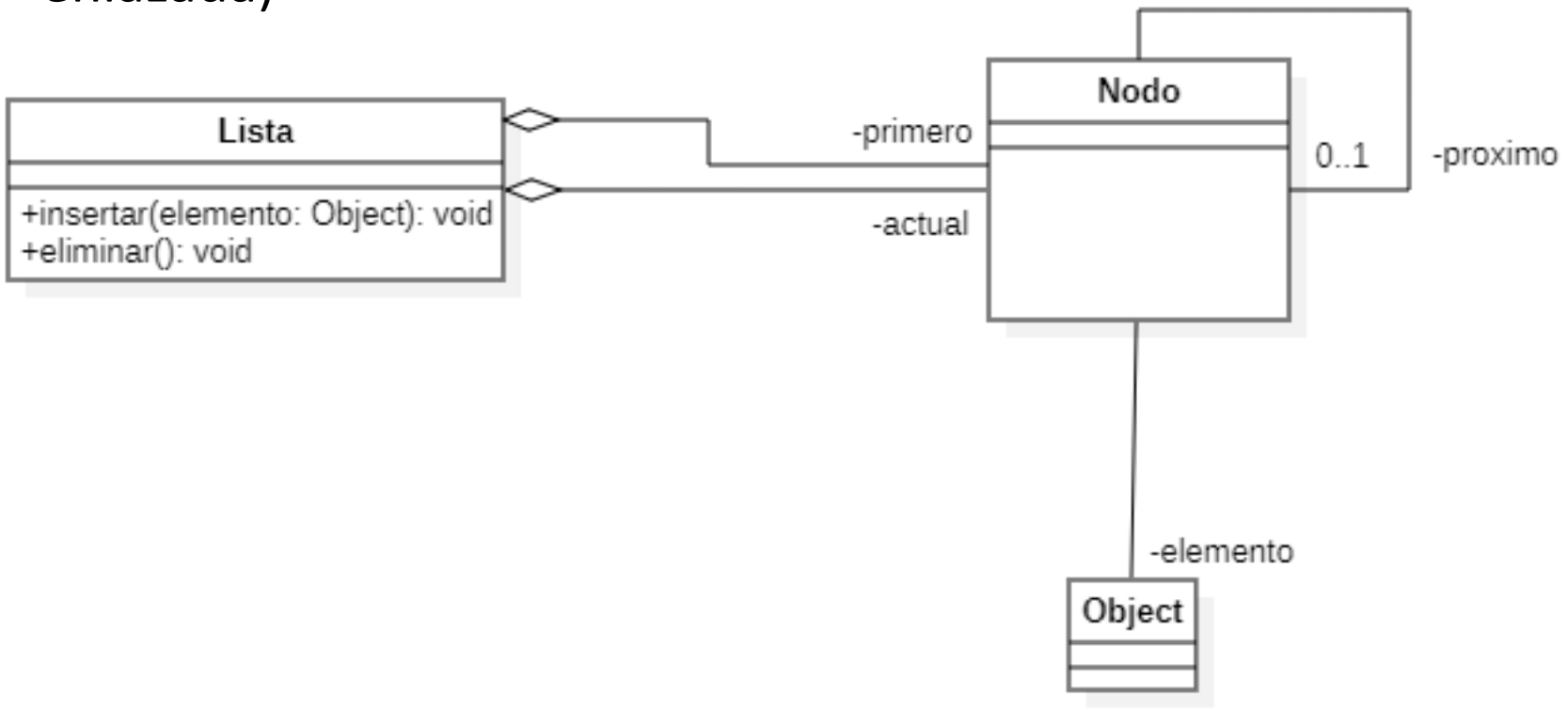
Asociaciones y Ensamblados en Java




```
class Clase1{
        Clase3 referencia;
}
class Clase2{
        Clase3 referencia;
}
class Clase3{
        Clase1 referencia[];
        Clase2 referencia[];
}
```

Práctica y Ejercicios

1. (Práctica)Pasar a código Java el diagrama UML de procesamiento de pedidos
2. (Ejercicio)Pasar el diagrama UML de abajo a código Java (Lista enlazada)



Lo que aprendimos

- Conceptos básicos de POO
- Vocabulario POO
- Relaciones entre clases
- Diseñar con UML