CIPFP Mislata
Centre Integrat Públic
Formació Professional Superior

UD05. DIAGRAMAS ESTRUCTURALES I:
DIAGRAMA DE CLASES

Entornos de Desarrollo
CFGS DAW

Curso 2018/2019

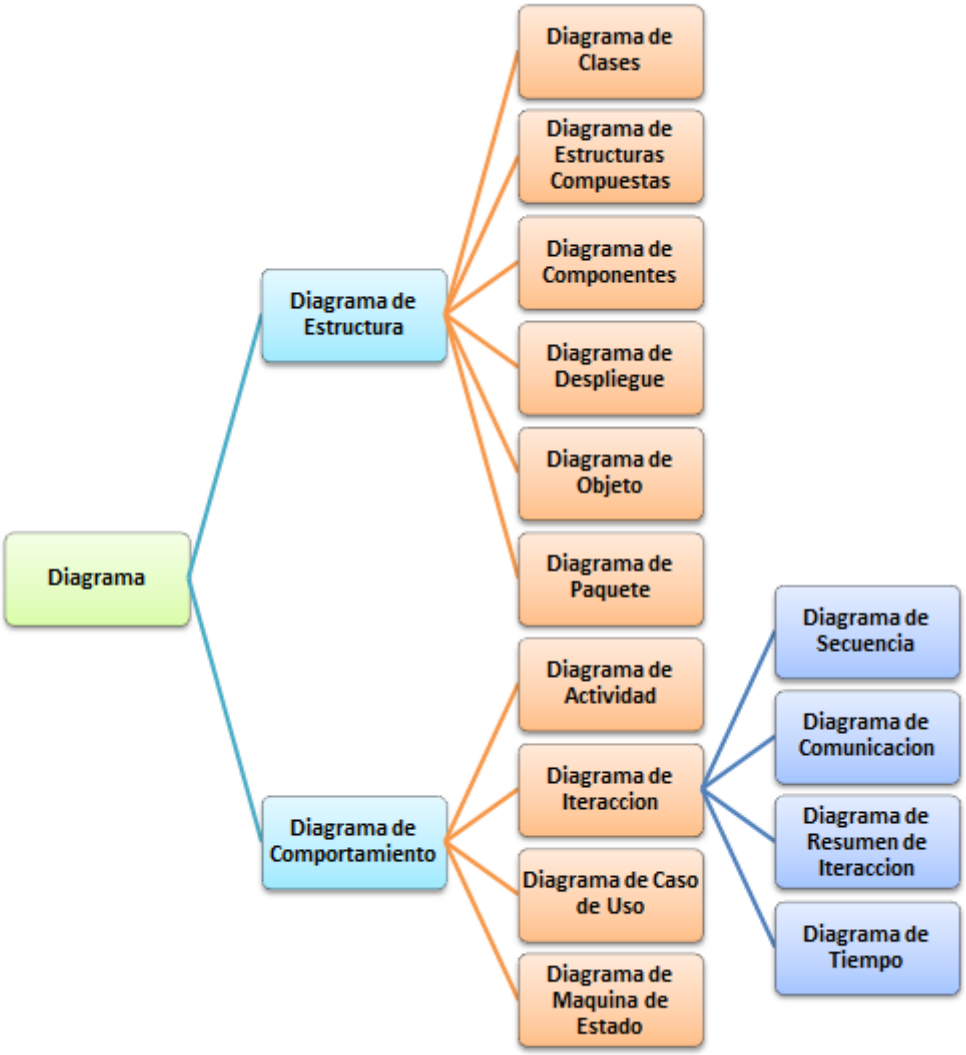
1 DIAGRAMAS ESTRUCTURALES

UML esta compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Al ser UML un lenguaje, existen reglas para combinar dichos elementos. En conjunto, los diagramas UML brindan diversas perspectivas de un sistema, Ahora bien, **el modelo UML describe lo que hará el sistema y no como será implementado**.

Una posible clasificación de los diagramas se puede hacer en función de la visión del modelo del sistema que ofrecen. Las dos visiones diferentes que pueden representar los diagramas UML son:

- **Visión estática (o estructural)**: Se utilizan objetos, atributos, operaciones y relaciones. La visión estática de un sistema d**a más valor a los elementos** que se encuentran en el modelo del sistema desde un punto de vista de la estructura del sistema. Describe aspectos del sistema que son estructurales y, por lo tanto, permanentes (**lo que el sistema tiene**)

- **Visión dinámica (o de comportamiento)**: Se **da más valor al comportamiento** dinámico del sistema, es decir, a lo que ha de pasar en el sistema. Con los diagramas de comportamiento se muestra como se modela la colaboración entre los elementos del sistema y sus cambios de estado (**lo que el sistema puede hacer**)



Dentro de los diagramas estáticos o estructurales se pueden encontrar 6 tipos de diagrama:

1. **Diagrama de clases**: Un diagrama de clases representa en un esquema gráfico las clases intervinientes y como se relacionan entre sí

2. **Diagrama de estructuras compuestas**: Muestra la **estructura interna** de una clase y las colaboraciones que esta estructura hace posibles

3. **Diagrama de componentes**: Es a la vez un diagrama de clases y de estructuras compuestas simplificado y más orientado a determinadas metodologías de programación

4 **Diagrama de despliegue**: Describe la configuración en tiempo de ejecución de un software especificado, normalmente, por un diagrama de componentes

5 **Diagrama de objetos**: Están vinculados con los Diagramas de Clases. Un objeto es una instancia de una clase, por lo que un diagrama de objetos puede ser visto como una instancia de un diagrama de clases. Los diagramas de objetos describen la estructura estática de un sistema en un momento particular y son usados para probar la precisión de los diagramas de clases

6 **Diagrama de paquetes**: representa esencialmente las relaciones de diferente tipo entre los contenidos de diferentes paquetes de un modelo

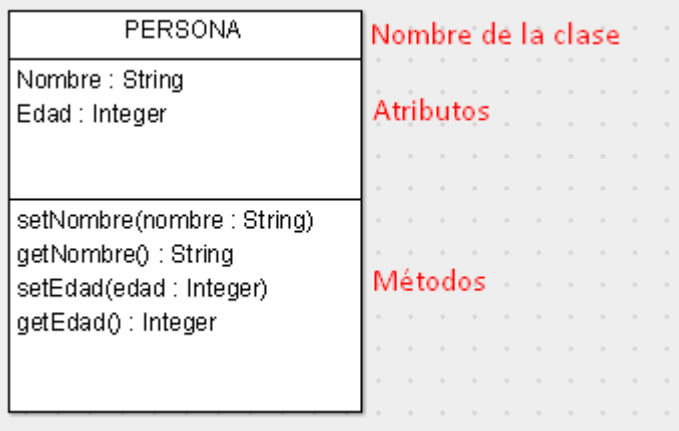
En este tema veremos los elementos y como crear un diagrama de clases.

2 DIAGRAMA DE CLASES

Los diagramas de clases muestran las diferentes clases que componen un sistema y cómo se relacionan unas con otras. Se dice que los diagramas de clases son diagramas **estáticos** porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: qué clases «conocen» a qué otras clases o qué clases «son parte» de otras clases, pero **no muestran los métodos mediante los que se invocan entre ellas**.

Este tipo de diagramas son utilizados durante las fases de **análisis y de diseño** de los proyectos de desarrollo del software. Es en estos momentos cuando se comienza a crear el modelo conceptual de los datos que utilizará el sistema. Por eso se **identifican** los **componentes** (con sus funcionalidades y atributos) que formarán parte en los procesos y se define las **relaciones** que habrá entre ellos.

Para representar las clases se utiliza un rectángulo dividido en tres zonas; la primera será el nombre de la clase, la segunda contendrá los atributos y la tercera los métodos.

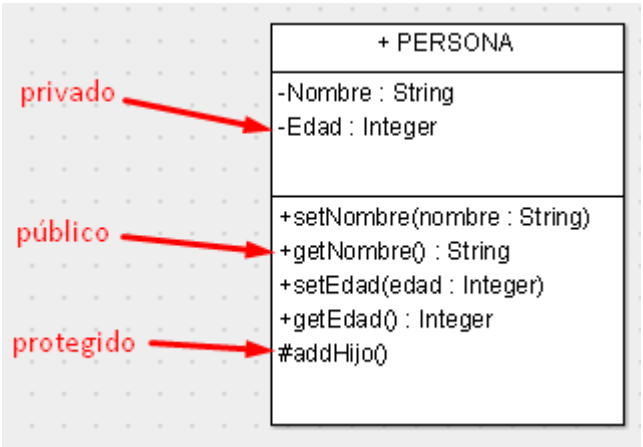


2.1 VISIBILIDAD

La visibilidad de un atributo o de un método definirá el **ámbito** desde el cual podrá ser utilizado este elemento. Esta característica está directamente relacionada con el concepto de orientación a objetos llamado **encapsulación**, mediante el cual se permite a los objetos decidir que información será más o menos pública para el resto de objetos.

Las posibilidades para la visibilidad, tanto de atributos como de métodos, son:

- **público** (+ en UML): El elemento será accesible para todos los otros elementos del sistema
- **privado** (- en UML): El elemento sólo será accesible para los elementos contenidos dentro del propio objeto
- **protegido** (# en UML): El elemento sólo será accesible para los elementos del propio objeto y para los elementos que pertenecen a objetos que son especializaciones del mismo



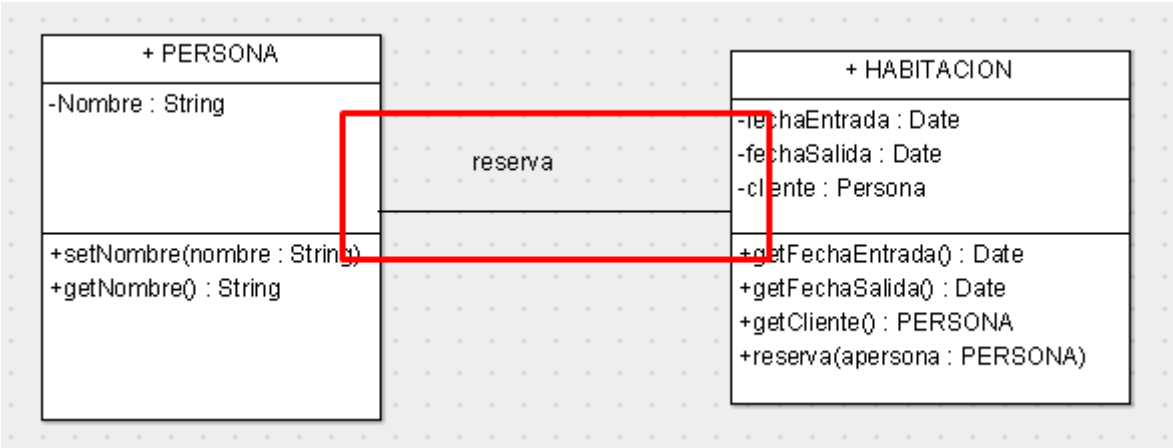
La clase anterior, codificada en Java quedaría:

```java
class Persona {

    private String nombre;
    private int edad;

    public void setNombre(String pnombre) {
        nombre = pnombre;
    }

    public void setEdad(int pedad) {
        edad = pedad;
    }

    public void getNombre() {
        return nombre;
    }

    public void getEdad() {
        return edad;
    }

    protected void addHijo(){

    }

}
```

2.2 RELACIONES

Las relaciones existentes entre las distintas clases nos indican como se comunican los objetos de esas clases entre sí; los mensajes *navegan* por las relaciones existentes entre las distintas clases.

Por ejemplo, si estamos creando una aplicación para gestionar la reserva de habitaciones de un hotel, tendremos dos clases llamadas *persona* y *habitación*:



La clase *persona* podrá ejecutar el método *reserva* de la clase *habitación* para reservar una habitación del hotel, con lo que se establecerá una relación entre ambas clases. :

Las relaciones pueden ser de distintos tipos:

- Asociación
- Herencia (Generalización/Especialización)
- Agregación/Composición
- Dependencia

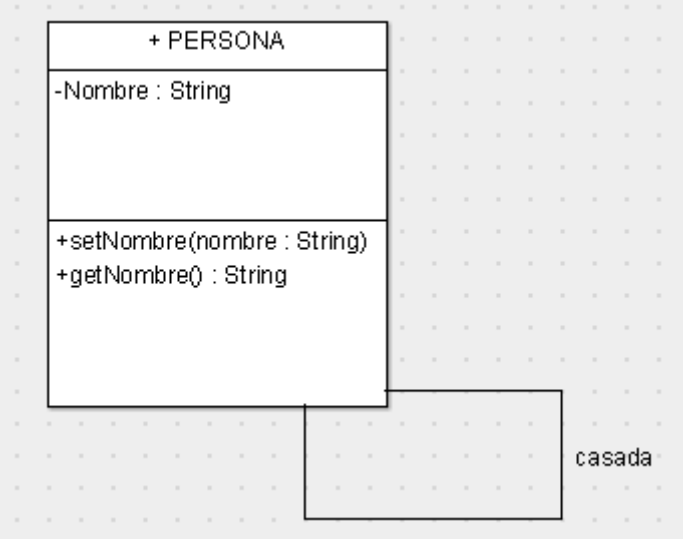
2.2.1 ASOCIACIÓN

Las relaciones que existen entre las diferentes clases se llaman de forma genérica **asociaciones**.

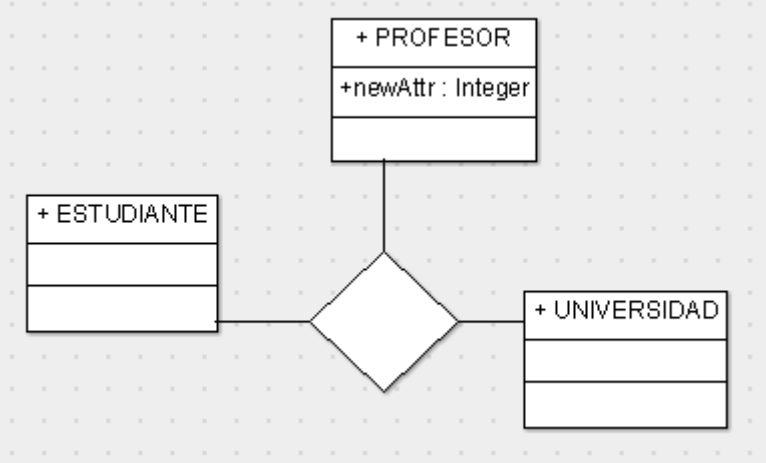
Gráficamente se representa como una línea continua que une las clases relacionadas entre sí. Las asociaciones deben nombrarse (normalmente con un verbo que indique la naturaleza de la relación que existe entre las clases).

Una asociación con dos extremos se dice que es **binaria**; sería el caso del ejemplo anterior (persona - *reserva* - habitación).

Cuando una clase se relaciona con ella misma, la asociación se dice que es **unaria** o **reflexiva**:



Una asociación con tres o más extremos se dice que es **N-aria**.

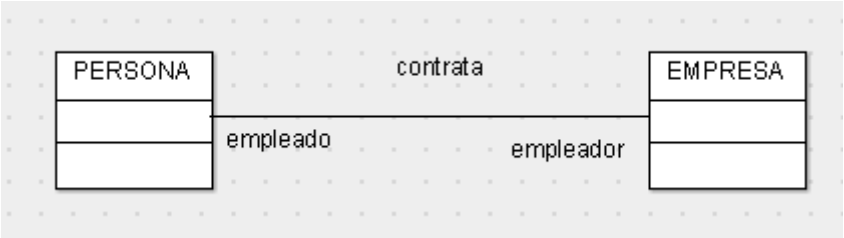


ROLES

Cada extremo de una asociación se denomina **rol**. Una asociación binaria tendrá dos roles. El nombre del rol indentifica el papel que el objeto juega en la relación.

Los roles son opcionales, y deben usarse si se necesitan para explicar mejor el diagrama.

Por ejemplo, en el siguiente ejemplo los roles *empleado* y *empleador* son los roles de las clases persona y empresa respectivamente en la relación contrata:

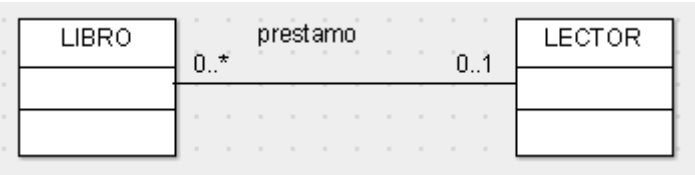


MULTIPLICIDAD

La **multiplicidad**, representada por unos valores [min...max], indica el número máximo de enlaces posibles que se pueden dar en una relación. Esta multiplicidad no podrá ser nunca negativa, siendo, además, el valor máximo siempre mayor o igual que el mínimo.

MULTIPLICIDAD	EJEMPLO	SIGNIFICADO
*	*	Cero o varios
1	1	Exactamente 1
[x]	3	Exactamente [x] (en este caso 3)
0..*	0..*	Cero o varios
0..1	0..1	Cero o uno
0..[x]	0..5	Cero o [x] (en este caso 5)
1..*	1..*	Uno o varios
1..[x]	1..7	Uno o [x] (en este caso 7)
[x]..[y]	2..8	Desde [x] hasta [y] (en este caso desde 2 hasta 8)

Por ejemplo, la siguiente asociación indica, en la asociación *prestamo*, que para cada objeto de la clase *Lector* podrá haber como mínimo 0 objetos de la clase *Libro* y como máximo n objetos. En cambio para cada objeto de la clase *Libro* podrá haber como mínimo 0 lectores y como máximo 1 objeto de la clase *Lector*.



Cuando la multiplicidad mínima es 0, la relación es opcional.

Una multiplicidad mínima mayor o igual a 1 establece una relación de obligatoriedad.

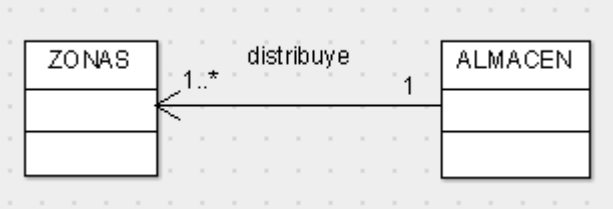
NAVEGABILIDAD

Una relación de asociación puede ser **unidireccional** o **bidireccional**, dependiendo de si ambas conocen la existencia la una de la otra o no.

La **navegabilidad** entre clases nos muestra que es posible pasar desde un objeto de la clase origen a uno o más objetos de la clase destino, dependiendo de la multiplicidad. En el caso de la asociación unidireccional la navegabilidad va en un sólo sentido, del origen al destino; el origen es navegable al destino, sin embargo, el destino no es navegable al origen.

Para representar una asociación unidireccional, se dibuja una flecha abierta en el extremo de la clase de objetos que son determinados.

Asociación bidireccional



Asociación unidireccional

En el caso de la asociación bidireccional, la clase *cliente* tendría que tener un set de objetos de la clase *facturas*, y ésta a su vez tendría un objeto de la clase *cliente*.

```
public class Cliente { //1 cliente tiene muchas facturas (utiliza HashSet)
    private HashSet<Facturas> facturas = new HashSet<Facturas>();
    public Cliente() { //constructor

    }

    public HashSet<Facturas> getFacturas() {
        return facturas;
    }

    public void setFacturas(HashSet<Facturas> newFacturas) {
        facturas = newFacturas;
    }
}

public class Facturas { //1 factura es de un cliente
    private Cliente cliente = null;
    public Facturas() { //constructor

    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente newCliente) {
        cliente = newCliente;
    }
}
```

En el caso de la asociación unidireccional, la clase *almacen* debería tener un set de objetos de la clase *zonas*. La clase *zonas* no tendría ningún objeto de tipo *almacen*, ya que la asociación va en el sentido *almacen->zonas*.

```
public class Zonas { //no sabe del a existencia de Almacen

    public Zonas() { //constructor

    }
}

public class Almacen { //1 almacén distribuye en muchas zonas (HashSet)

    private HashSet<Zonas> zonas = new HashSet<Zonas>();

    public Almacen() { //constructor

    }

    public HashSet<Zonas> getZonas() {
        return this.zonas;
    }

    public void setZonas(HashSet<Zonas> newZonas) {
        this.zonas = newZonas;
    }
}
```

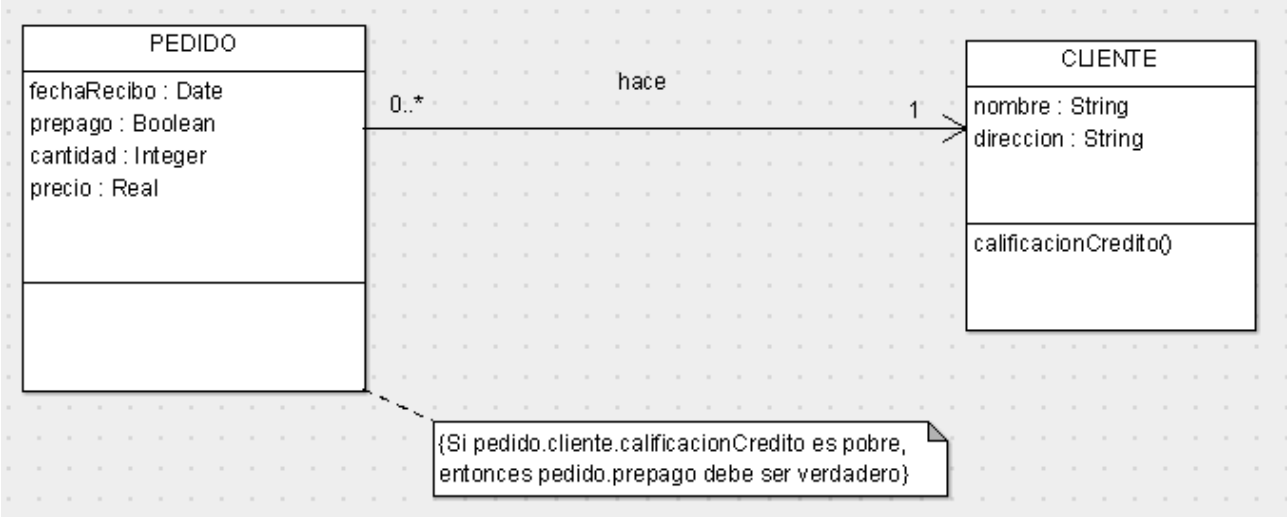
RESTRICCIONES

Un diagrama de clases está expresando (a través del dibujo) restricciones que deben cumplir las clases.

De este modo, la cardinalidad sería una restricción entre clases. Por ejemplo, una factura sólo puede pertenecer a un cliente.

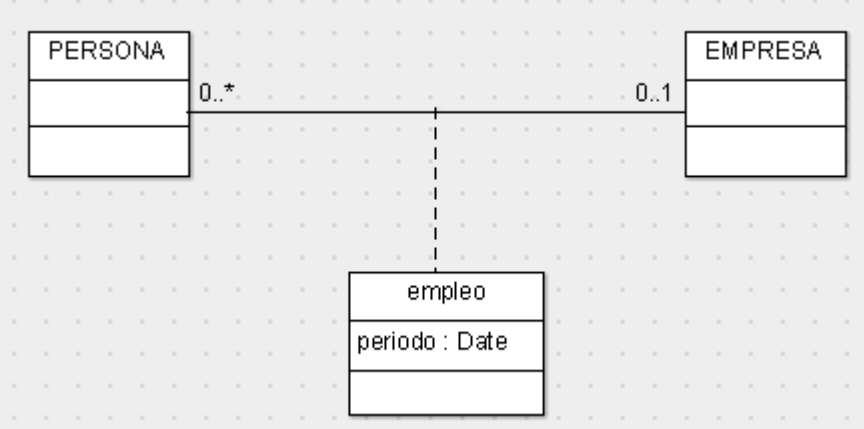
Pero hay veces que el diagrama no expresa todos los requisitos. Para ello se utiliza las **restricciones de integridad**.

UML no define una sintaxis estricta para describir las restricciones, aparte de ponerlas entre llaves ({}). Se pueden escribir en lenguaje informal para simplificar su lectura. Lo ideal sería escribir un fragmento de código de programación si ya sabemos con que lenguaje se va a programar.



CLASE ASOCIATIVA

Cuando una asociación tiene propiedades o métodos propios se representa como una clase unida a la línea de la asociación por medio de una línea discontinua. Tanto la línea como el rectángulo de clase representa el mismo elemento conceptual: la asociación.



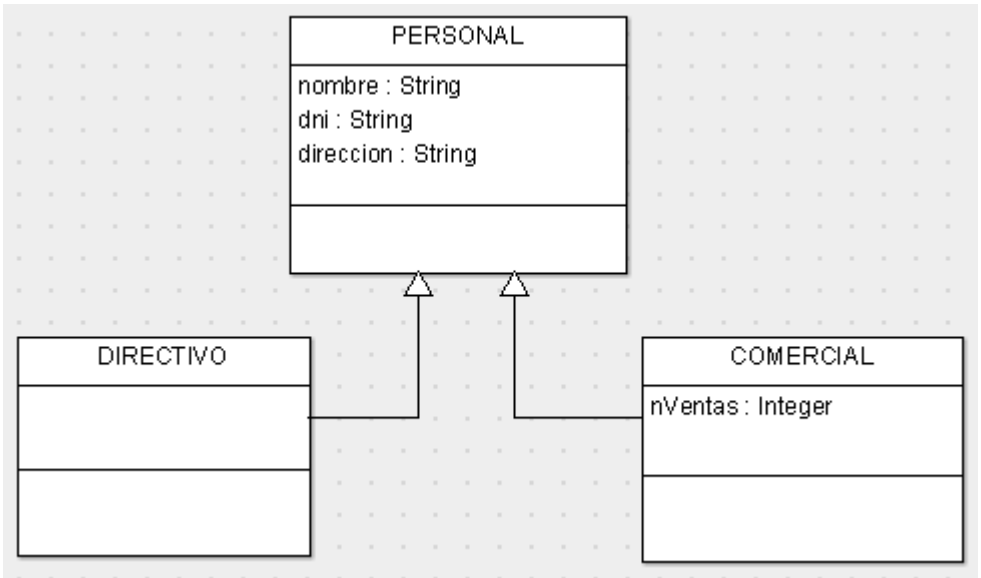
2.2.2 HERENCIA

La **relación de generalización** se da entre dos clases donde hay un vínculo que se puede considerar de herencia.

Una clase se llama madre o superclase. La otra (o las otras) son las llamadas clases hijas o subclases, que heredan los atributos y los métodos y comportamiento de la clase madre.

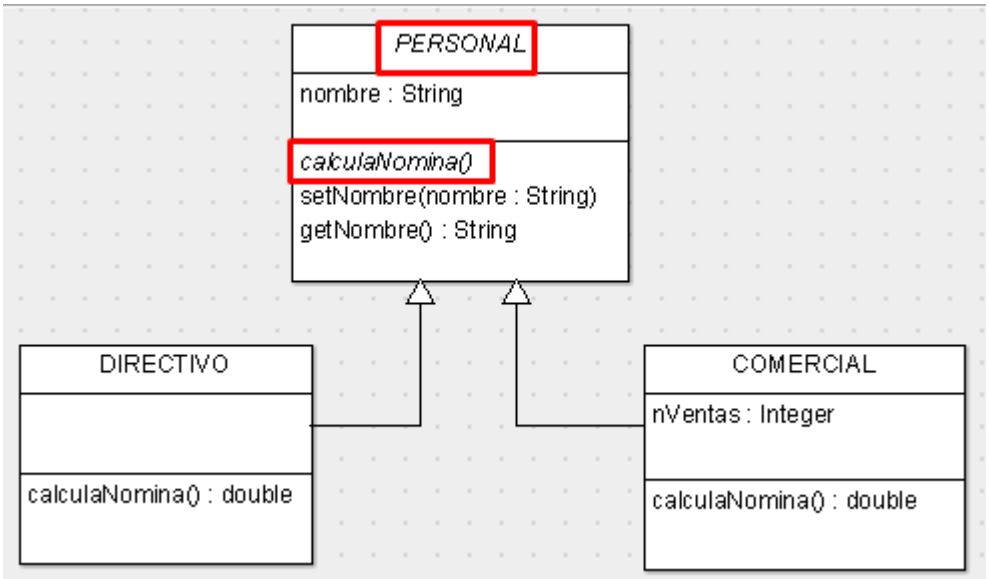
Este tipo de relación se representa mediante una flecha que sale de la clase hija y que acaba en la clase madre.

La herencia ofrece como punto fuerte la posibilidad de reutilizar parte del contenido de un objeto (la superclase) extendiendo sus atributos y sus métodos al objeto hijo.



CLASES ABSTRACTAS

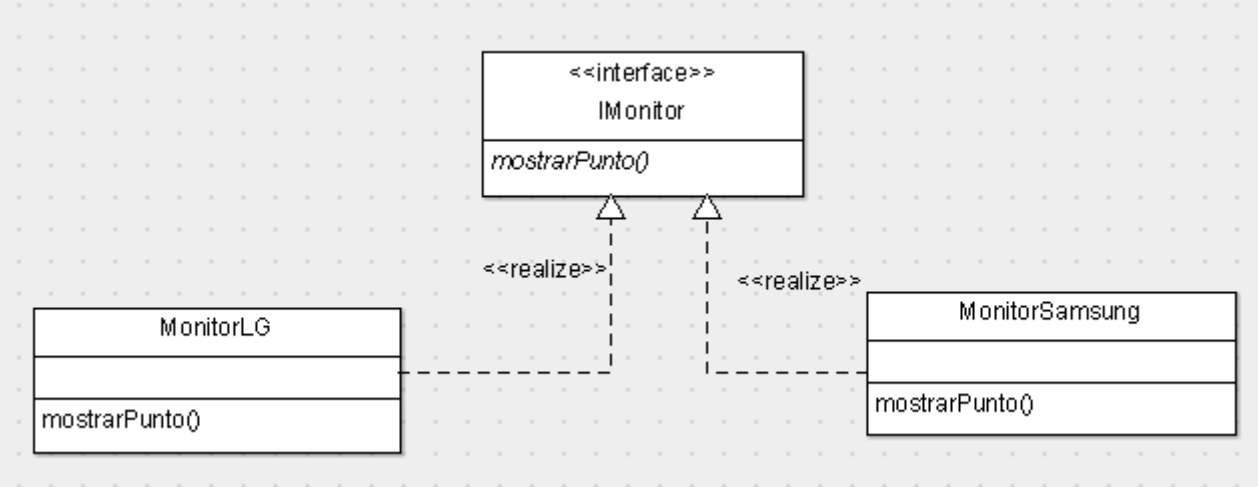
Una clase abstracta se representa con el nombre de la clase y sus métodos abstractos en cursiva. Esto indica que la clase definida no puede ser instanciada, pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es definiendo subclases que implementen los métodos abstractos definidos. La clase abstracta obliga a que haya una herencia.



En este ejemplo, la clase *personal* es una clase abstracta ya que tiene el método abstracto *calculaNomina()*. Las subclases que hereden de *personal* deberán implementar cada una el método calculaNomina() (el nombre del método en las subclases ya no irá en cursiva, ya que en las clases *directivo* y *comercial* no será un método abstracto).

INTERFACES

La asociación entre una clase y su interfaz se representa mediante una flecha cerrada con línea punteada, cuya punta va en dirección al interfaz. La flecha se etiqueta con la palabra *realize*.



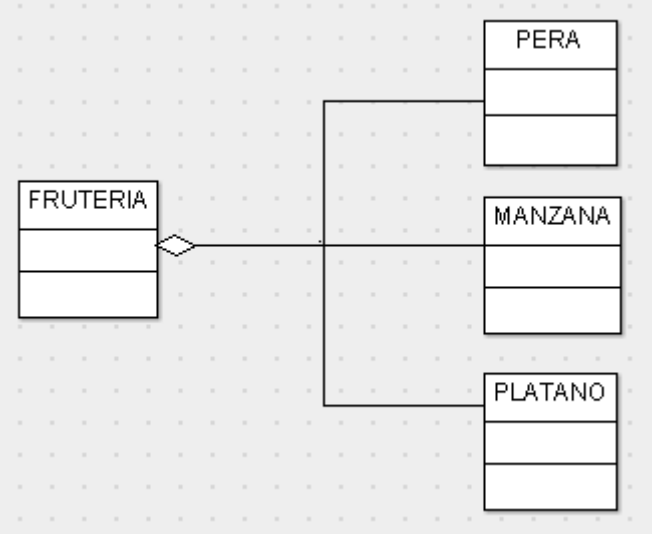
En este ejemplo creamos un interfaz llamado *IMonitor*, con un método abstracto (recordar que, por definición, todos los métodos de un interfaz tienen que ser abstractos) *mostrarPunto()*. Las dos subclases *MonitorLG* y *MonitorSamsung* deberán implementar cada una su propio método *mostrarPunto()*.

2.2.3 AGREGACIÓN/COMPOSICIÓN

Una **relación de asociación de agregación** es un caso especial entre dos o más objetos. Se trata de una relación del tipo **todo-parte**.

Este tipo de relaciones implica dos tipos de objetos. El objeto llamado **base** y el objeto que estará incluido en el objeto base. La relación indica que el objeto base necesita del objeto incluido para poder existir y llevar a cabo sus funcionalidades.

Si desaparece el objeto base, el o los objetos que se encuentran incluidos en el objeto base no desaparecen y podrán continuar existiendo con sus funcionalidades propias. La relación de asociación de agregación se representa mediante una línea continua que finaliza en uno de los extremos por un rombo. El rombo se ubicará en la parte del objeto base.

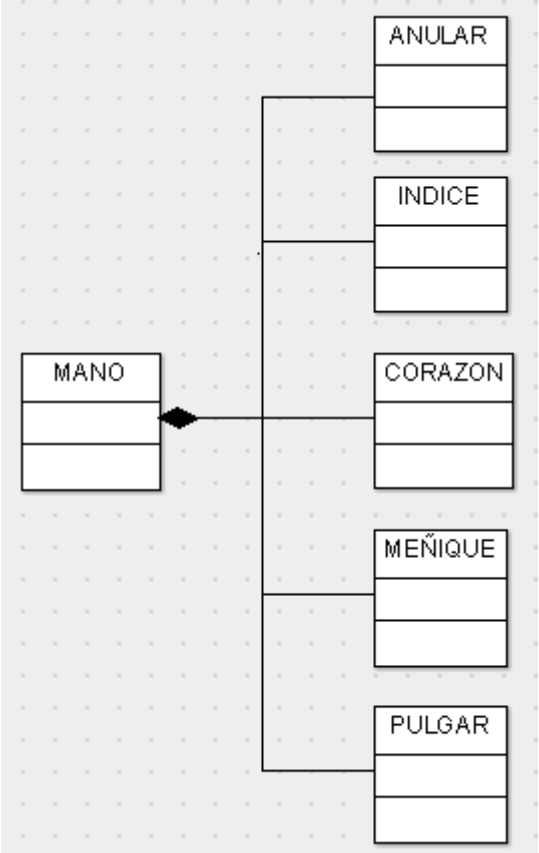


Una **relación de asociación de composición** es también un caso especial de asociación entre dos o más objetos. Es una relación del tipo **todo-parte**.

Es una relación muy parecida a la relación de asociación de agregación, con la diferencia de que hay una relación de existencia entre el objeto base y el objeto (o los objetos) que están incluidos. Es decir, **si el objeto base deja de existir, los objetos incluidos también dejarán de existir**.

El tiempo de vida de los objetos incluidos dependerá del tiempo de vida del objeto base.

La relación de composición se representa mediante una línea continua finalizada en un extremo por un rombo pintado. El rombo pintado se ubicará en la parte del objeto base.



2.2.4 DEPENDENCIA

Otro tipo de relación entre clases es la **relación de dependencia**. Este tipo de relación se representa mediante una flecha discontinua entre dos elementos.

El objeto del que salga la flecha se considera un objeto dependiente. El objeto al que llegue la flecha se considera un objeto independiente.

Se trata de una relación semántica. Si hay un cambio en el objeto independiente, el objeto dependiente se verá afectado.