TEMA 5: Diagramas de clase

1.- Introducción

OBJETIVO:

- Identificar los conceptos básicos de la programación orientada a objetos.
- Elaborar e interpretar diagramas de clase sencillos.
- Utilizar herramientas para desarrollar diagramas de clases
- Generar código a partir de diagramas de clases
- Realizar ingeniería inversa a partir de código.

Para el análisis y diseño orientado a objetos se utiliza UML (Lenguaje de Modelado Unificado). Es un lenguaje de modelado basado en diagrama que sirve para expresar modelos (un modelo es una representación de la realidad donde se ignora detalles de menor importancia).

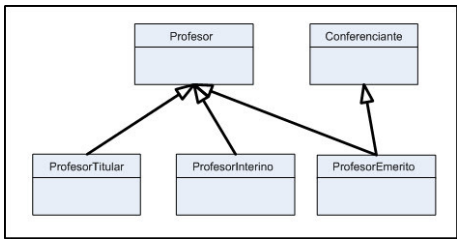
2.- Conceptos orientados a objetos

Un objeto es aquello que tiene estado (propiedades más valores) , comportamiento (acciones y reacciones a mensajes) e identidad (propiedades que lo distingue de los demás objetos). La estructura y comportamiento de objetos similares están definidos en su clase común; los términos y objeto son intercambiables. Una clase es un conjunto de objetos que comparten una estructura y comportamiento común.

En la programación orientada objetos (POO) los principíos del modelado son: abstracción, encapsulación, modularidad, jerarquía y polimorfismo y en menor medida tipificación /typing), concurrencia y persistencia.

- **Abstracción.** Denota las características esenciales de un objeto , donde se capturan sus comportamientos, El objetivo es obtener una descripciòn formal . La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a componer un conjunto de clases que permitan modelar la realidad o el problema que se quiere resolver.
- **Encapsulación.** Es el proceso de ocultar todos los detalles de un objeto que no contribuye a sus características esenciales. La encapsulación consiste en ocultar los atributos y métodos del objeto a otros objetos, estos no deben estar expuestos a los objetos exteriores . Una vez encapsulados, pasan a denominarse atributos y métodos privados del objeto.
- **Modularidad**. Es la propiedad de una aplicación o de un sistema que ha sido descompuesto en un conjunto de módulos o partes más pequeñas coherentes e indepentientes.
- **Jerarquía o herencia.** La POO introduce la posibilidad de extender clases, produciendo nuevas definiciones de clases que heredan todo el comportamiento y código de la clase extendida. La clase original se denomina clase padre o superclase. La nueva clase que se define como una extensión se denomina clase hija o subclase. La extención de una clase se llama herencia, porque la clase hija hereda todos los métodos y atributos de la clase padre que se extiende . Cada subclase estaría formada por un grupo de objetos más especializados con características comunes que compartirían datos y operaciones. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

Java no permite la herencia multiple pero a cambio dispone de la construcción denominada "Interface" que **permite** una forma de simulación o implementación limitada de la **herencia múltiple**.



- **Polimorfismo**. Consiste en reunir con el mismo nombre comportamientos diferentes.Es la propiedad por la cuál un mismo mensaje puede originar conductas completamente diferentes al ser recibido por diferentes objetos. Dos instancias u objetos , pertenecientes a distintas clases , pueden responder a la llamada a métodos del mismo nombre, cada uno de ellos con distinto comportamiento encapsulado, pero que responden a una interfaz común .

Ejemplo de polimorfismo.

1. Supongamos dos clases distintas ; la clase Gato y la clase Perro , que heredan de la clase padre Animal. La clase Animal tiene el método abstracto emiteSonido() que se implementa de forma distinta en cada una de las subclases (gato y perro suenan de forma distinta). Entonces , un tercer objeto puede enviar el mensaje de hacer sonido a un grupo de objetos Gato y Perro por medio de una variable de referencia de clase Animal, haciendo así un uso polimórfico de dichos objetos respecto del mensaje mover.

```
class Animal {
  public void emiteSonido() {
    System.out.println("Grr...");
  }
}
class Cat extends Animal {
  public void emiteSonido() {
    System.out.println("Miauuu");
  }
}
class Dog extends Animal {
  public void emiteSonido() {
    System.out.println("Guauuu");
  }
}
```

Como todos los objetos Gato y Perro son objetos Animales, podemos hacer lo siguiente

```
public static void main(String[ ] args) {
  Animal Pluto = new Dog();
  Animal Suli = new Cat();
}
```

Creamos dos variables de referencia de tipo Animal y las apuntamos a los objetos Gato y Perro. Ahora, podemos llamar a los métodos emiteSonido().

```
Pluto.emiteSonido();
//Outputs "Guauuu"

Suli.emiteSonido();
//Outputs "Miauuuu"
```

Por lo general diremos que existen **3 tipos de polimorfismo**:

- **Sobrecarga**: El más conocido y se aplica cuando existen funciones con el mismo nombre en clases que son completamente independientes una de la otra.
- **Paramétrico**: Existen funciones con el mismo nombre pero se usan diferentes parámetros (nombre o tipo). Se selecciona el método dependiendo del tipo de datos que se envíe.
- **Inclusión**: Es cuando se puede llamar a un método sin tener que conocer su tipo, así no se toma en cuenta los detalles de las clases especializadas, utilizando una interfaz común.

- **Tipificación.** Es la definición precisa de un objeto , de tal forma que objetos de diferentes tipos no puedan ser intercambiados o , como mucho, podrán intercambiarse de manera muy restringida.
- **Concurrencia.** Es la propiedad que distingue un objeto que está activo de uno que no lo está. El objeto activo está haciendo algo, se utilizan sobre todo en la programación concurrente o multihilo.
- **Persistencia**. Es la propiedad de un objeto a través de la cual su existencia trasciende el tiempo (es decir , el objeto continua existiendo después de que su creador ha dejado de existir). Se refiere a objetos de clases asociadas a Base de Datos Orientadas a Objetos o a Bases de Datos Objeto Relacionales.

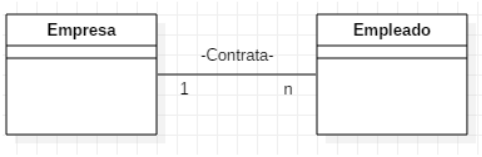
3.- UML (Lenguaje de modelado unificado)

Es un lenguaje gráfico para visualizar , especificar y documentar cada una de las partes que comprende el desarrollo de software. Podemos decir que UML es un lenguaje que se utiliza para documentar .

3.1.- Tipos de diagrámas

Cada diagrama UML representa alguna parte o punto de vista del sistema. Los diagramas más utilizados son los siguientes:

- **Diagramas de clase.** Muestran las diferentes clases que componen un sistema y cómo se relacionan unas con otras



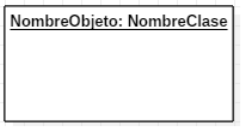
Ejemplo de relación Empresa-Empleado

En la figura anterior se muestran las clases Empresa y Empleado , y su relación o asociación. Una empresa tiene n empleados y un empleado trabaja en una empresa. Un objeto Empresa tendrá muchos objetos empleados.

- **Diagramas de objetos**. Representan objetos y sus relaciones. Muestra una serie de objetos (instancias de las clases) y sus relaciones en un momento particular de la ejecución del sistema. Es un diagrama de instancias de las clases mostradas en el diagrama de clases.

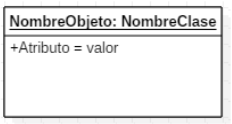
  El diagrama de objeto se compone de los siguientes elementos:

  a) **Objetos**. Cada objeto se representa con un rectángulo con su nombre y el de su clase



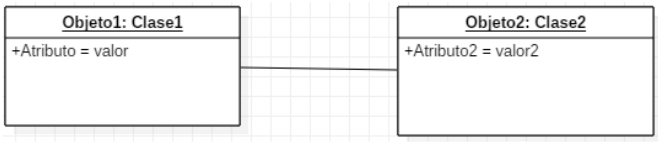
Representación de un objeto

  b) **Atributos**. A diferencia de las clases , los atributos pueden tener valores asignados.



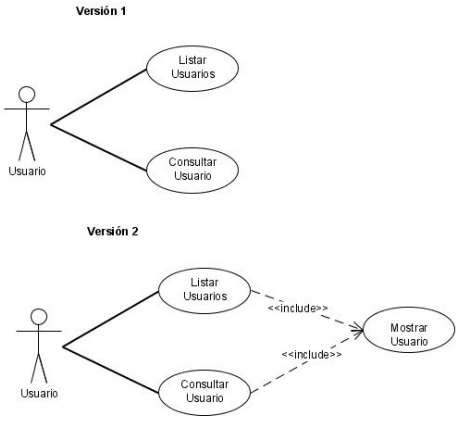
Representación de un atributo

  c) **Vinculos:** Son asociaciones entre dos objetos y se representan con los mismos elementos que en el diagrama de clase.



Asociación entre dos objetos

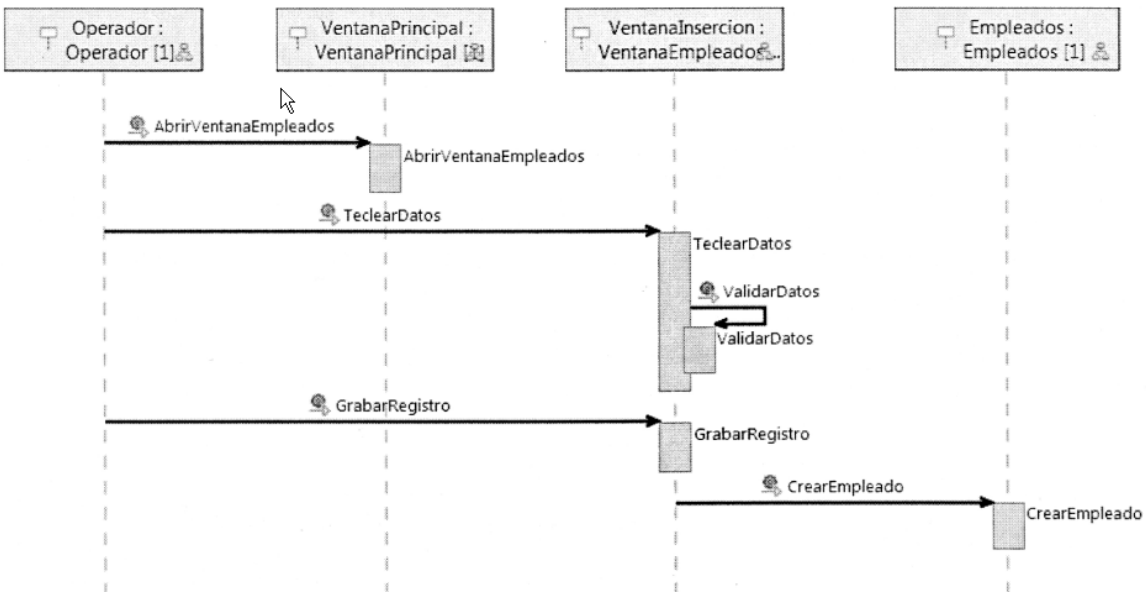
- **Diagramas de casos de uso**. Se utiliza para entender el uso del sistema, muestran un conjunto de actores , las acciones (casos de uso) que se realizan en el sistema, y las relaciones entre ellos.



En la figura anterior se muestra dos ejemplos de diagrama de uso en el que el actor usuario realiza consultas sobre los datos de los usuarios de una base de dato.

- **Diagrama de secuencia**. Son una representación temporal de los objetos y sus relaciones.

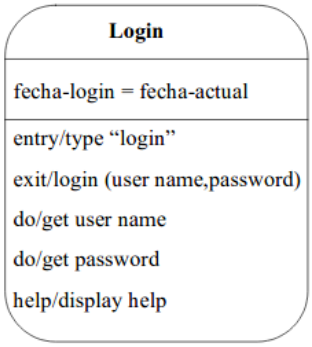
En el siguiente ejemplo, se muestra el diagrama de secuencia de la inserción de datos de un empleado en la base de datos. El operador indica a la ventana principal (Tenemos una aplicación con una ventana principal que da paso a las ventanas para poder operar con empleados y departamentos) que quiere realizar operaciones con empleados, y pide abrir la ventana de empleados. En la ventana de empleados se teclearán los datos, se validarán los datos , y el operador indicará si se graba el registro en la BD . La orden de grabar el registro creará un nuevo objeto Empleado en la base de datos con los datos tecleados.



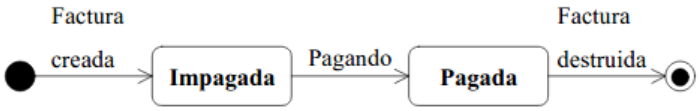
En el diagrama de secuencia anterior, se muestra la secuencia de inserción de datos de un empleado en la base de datos. El operador indica a la ventana principal que quiere realizar operaciones con empleados, y pide abrir la ventana de empleados. En la ventana de empleados se teclean los datos , se validarán los datos, y el operador indicará si se graba el registro en la BD. La orden de grabar el registro creará un nuevo objeto Empleado en la base de datos con los datos tecleados

- **Diagrama de estado.** Se utiliza para analizar los cambios de estado de los objetos. Muestra los estados, eventos, transiciones y actividades de los diferentes objetos.

Un estado se representa gráficamente por medio de un rectángulo con los bordes redondeados y con tres divisiones internas. Los tres compartimentos alojan el nombre del estado, el valor característico de los atributos del objeto en ese estado y las acciones que se realizan en ese estado, respectivamente. En muchos diagramas se omiten los dos compartimentos inferiores.

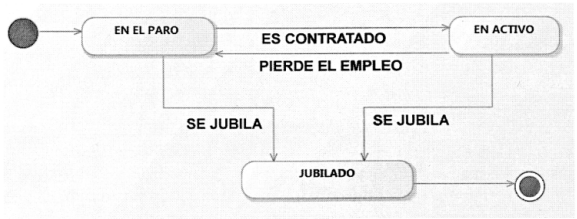


En esta figura, se representa el estado Login junto con sus tres divisiones .



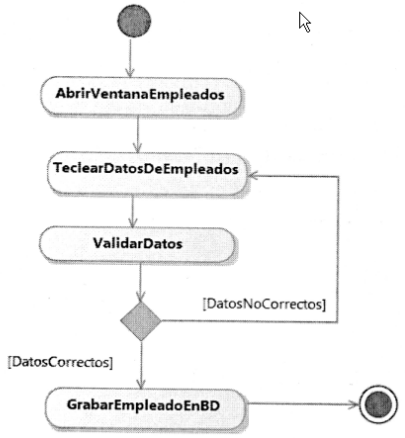
Los diagramas de estado tienen un punto de comienzo , el estado inicial, que se dibuja mediante un círculo sólido , y uno (o varios) punto de finalización, el estado final , que se dibuja por medio de un círculo conteniendo otro más pequeño y relleno.

En la siguiente imagen tenemos que un trabajador cuando es contratado pasa al estado EN ACTIVO, si pierde el empleo pasa al estado EN EL PARO . Si se jubila pasa al estado JUBILADO.



- **Diagramas de actividad.** En UML , un diagrama de actividad se utiliza para mostrar la secuencia de actividades. Los diagramas de actividades muestran el flujo de trabajo desde un

punto de inicio hasta el punto final detallando las decisiones que surgen en la progresión de eventos contenidos en la actividad.



Ejemplo de la inserción de un empleado en la BD.

- **Diagrama de despliegue**. Especifica el hardware físico sobre el que el sistema de software se ejecutará y también especifica cómo el software se despliega en ese hardware. Está compuesto de nodos. Un nodo es una unidad material capaz de recibir y de ejecutar software. La mayoría de nodos son ordenadores

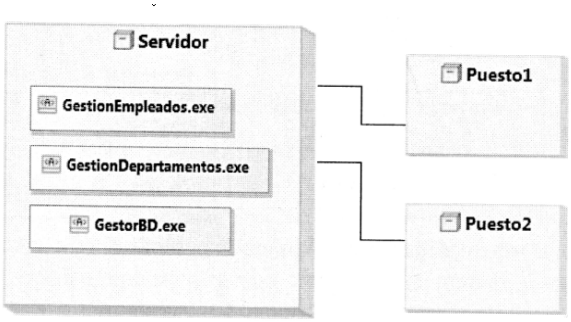
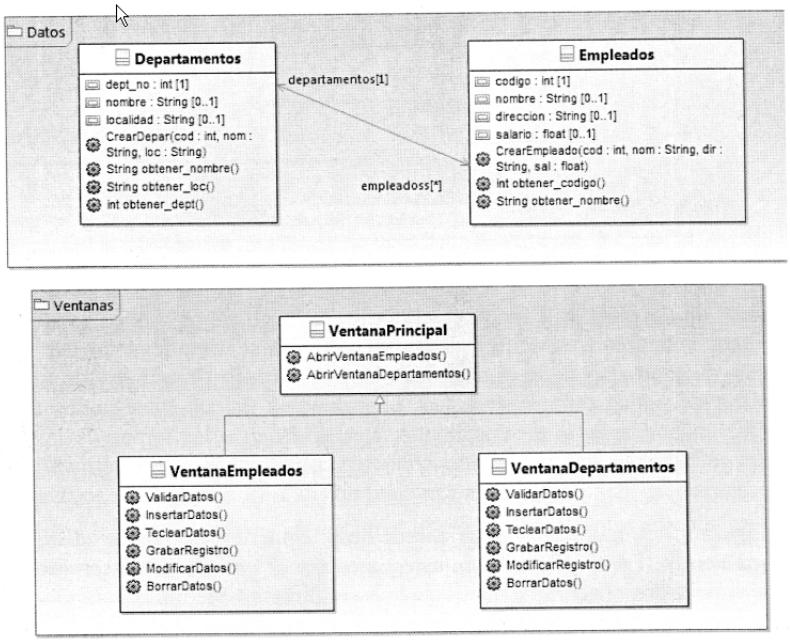


Diagrama de despliegue del sistema de gestión de empleados y departamentos. La arquitectura de este sistema está basada en un servidor y dos puestos clientes conectados al servidor mediante enlace directos que representa la red. El servidor tiene 3 ejecutables , el encargado de la gestión de empleados, el encargado de la gestión de departamentos y el gestor de la BD.

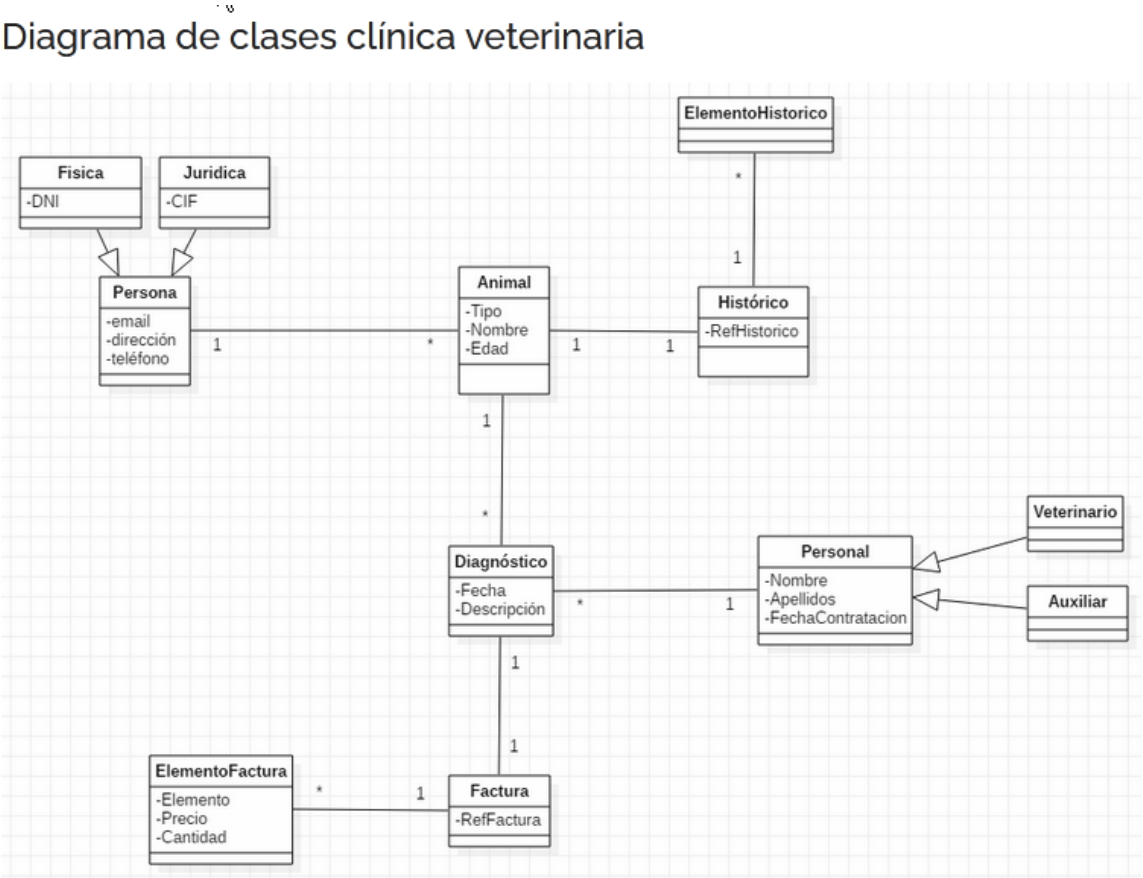
- **Diagramas de paquetes.** Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Cuando se utiliza para representar elementos de clase, los diagramas de paquetes proporcionan una visualización de los espacios de nombres. El uso más común de los diagramas de paquetes es organizar diagramas de casos de uso y diagrama



En esta imagen, tenemos un diagrama de clases formado por 5 clases, las clases Departamentos y Empleados agrupadas en el paquete Datos, y las clases VentanaPrincipal , VentanaEmpleado y VentanaDepartamento agrupadas en el paquete ventanas

4.- Diagramas de clases

Un diagrama de clases describe la estructura de un sistema mostrando sus clases y las asociaciones entre ellas. Sirve para visualizar las relaciones entre las clases que componen el sistema.



4.1.- Elementos de un diagrama de clases.

Está compuesto por los siguientes elementos:

- Clases: atributos, métodos y visibilidad
- Relaciones e interfaces: asociación, herencia, agregación, composición, realización y dependencia

4.1.1.- Clases

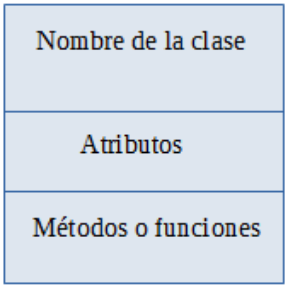
Las clases son la unidad que encapsula toda la información de un objeto. A través de ella podemos modelar el entorno en estudio (un empleado, un departamento, un artículo, etc).

La forma más rápida de encontrar clases sobre un enunciado , sobre una idea de negocio o, en general , sobre un tema concreto es ***buscar los sustantivos*** que aparecen en el mismo.

Ejemplo, algunas clases podrían ser: Animal, Persona, Mensaje, Expediente, ….. Es un concepto muy amplio y resulta fundamental identificar de forma efectiva estas clases, porque de no hacerlo podemos encontrarnos posteriormente con problemas que nos obliguen a realizar un nuevo análisis.

4.1.1.1.- Componentes de una clase

Una clase está formada por el nombre, los atributos y los métodos o funciones.



- La primera de las zonas se utiliza para el **nombre de la clase**. En caso de que la clase sea **abstracta** se utilizará su nombre en cursiva.

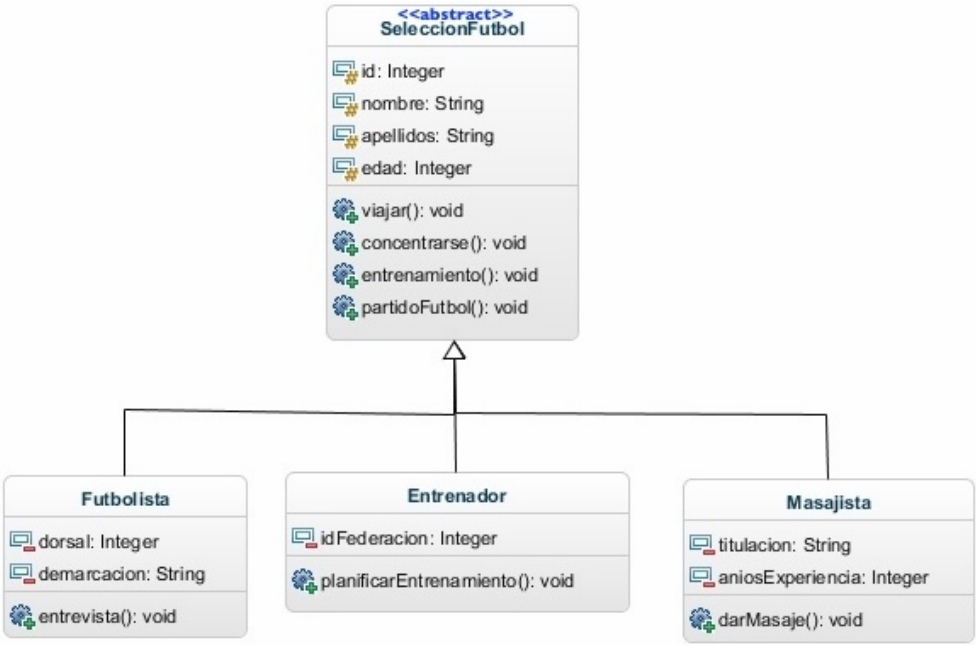
- La segunda de las zonas se utiliza para escribir los **atributos** de la clase, uno por línea y utilizando el siguiente formato:

visibilidad nombre_atributo : tipo = valor-inicial { propiedades }

- La última de las zonas incluye cada una de las **funciones** que ofrece la clase. De forma parecida a los atributos, sigue el siguiente formato:

visibilidad nombre_funcion { parametros } : tipo-devuelto { propiedades }

De la misma manera que con los atributos, se suele simplificar indicando únicamente el nombre de la función y, en ocasiones, el tipo devuelto.



Atributos

Representa alguna propiedad de la clase que se encuentra en todas las instancias de la clase. Ejemplo de atributos son: Nombre, Salario, Teléfono, etc. Al crear los atributos se indicará el tipo de dato, los tipos básicos UML son Integer, String y Boolean. Al crear el atributo se indicará su visibilidad con el entorno, la visibilidad está estrechamente relacionada con el encapsulamiento y esta puede ser:

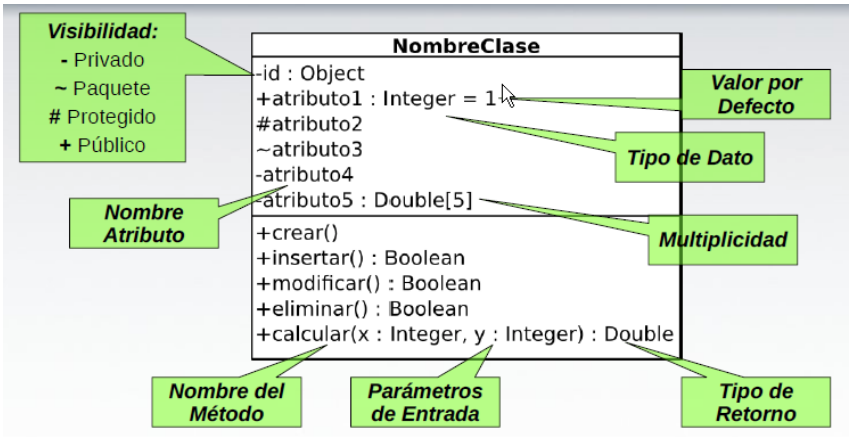
- **(+) Pública.** Representa que se puede acceder al atributo desde cualquier lugar de la aplicación, es decir se puede acceder tanto dentro como fuera de la clase.
- **(-) Privada.** Representa que se puede acceder al atributo únicamente desde la misma clase.
- **(#) Protegida.** Representa que el atributo puede ser accedido únicamente desde la misma clase o desde las clases que hereden de ella (clases derivadas).
- **(~) Pakage**. El atributo empaquetado es visible en las clases del mismo paquete.

Publico (public)	+	Elemento no encapsulado visible para todos
Protegido (protected)	#	Elemento encapsulado visible en la clase y en sus subclases
Privado (private)	-	Elemento encapsulado visible solo en la clase
Paquete (pakage)	~	Elemento encapsulado visible solo en las clases del mismo paquete

Métodos

Un método , es la implementación de un servicio de la clase que muestra un comportamiento común a todos los objetos. Define como interactúa con su entorno. Los métodos pueden ser:

- **(+) public.** Representa que se puede acceder al método desde todos lados.
- **(-) private.** Solo los métodos de la misma clase pueden acceder a él.
- **(#) protected.** El método puede ser accedido por métodos de la clase además de los métodos de las subclases que se derivan.
- **(~) pakage**. El método solo es visible en las clases del mismo paquete.



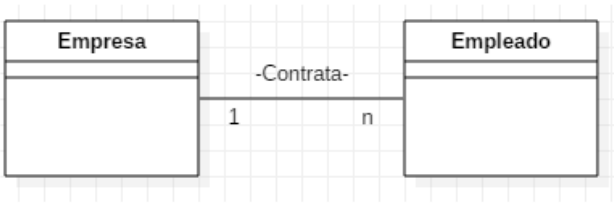
4.1.2.- Relaciones

Una relación i**dentifica una dependencia**. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina *dependencia reflexiva*. Por ejemplo , la relación existente entre un alumno y el curso en el que está matriculado, la relación entre un profesor y el centro en el que trabaja. En UML las relaciones se describen mediante asociaciones, de igual modo que los objetos se describen mediante clases.
Las relaciones se representan con una linea que une las clases, esta línea variará dependiendo del tipo de relación

Multiplicidad. Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango… Se utiliza *n* o * para identificar un número cualquiera.

Notación	Cardinalidad / Multiplicidad
0 . . . 1	Cero o una vez
1	Una y solo una vez
*	De cero a varias veces
0 … *	De cero a varias veces
1 … *	De una a varias veces
M … N	Entre M y N veces
N	N veces

Nombre de la relación. En ocasiones se escriba una indicación de la relación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como por ejemplo: “Una empresa contrata a n empleados”



Ejemplo de relación Empresa-Empleado

4.1.2.1.- Tipos de relaciones

- **Asociación.**
- **Agregación.**
- **Composición.**
- **Dependencia.**
- **Herencia.**

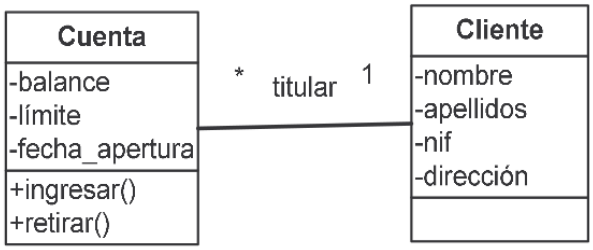
Asociación

Es una relación estructural que describe una conexión entre objetos. Dentro de una relación de asociación , cada clase juega un rol , que se indica en la parte superior o inferior de la línea que conecta a dichas clases.

Aunque las asociaciones suelen ser bidireccionales (se pueden recorrer en ambos sentidos), en ocasiones es deseable hacerlas unidireccionales (restringir su navegación en un único sentido).

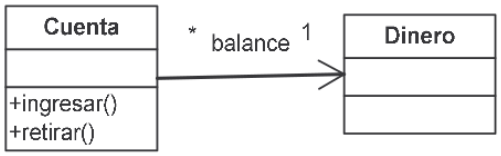
Si se convierte a Java dos clases unidas por una asociación bidireccional , cada una de las clases tendrá un objeto o un set de objeto, dependiendo de la multiplicidad. En el caso de la asociación Unidireccional la navegabilidad va en un solo sentido , del origen al destino; el origen es navegable al destino, sin embargo , el destino no es navegable al origen.

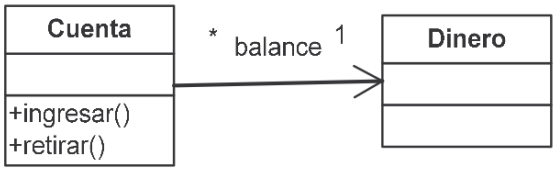
Relación bidireccional



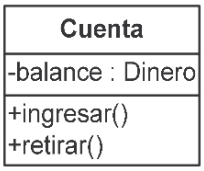
En esta asociación bidireccional, un cliente puede tener varias cuentas y la cuenta es de un cliente

Relación unidireccional





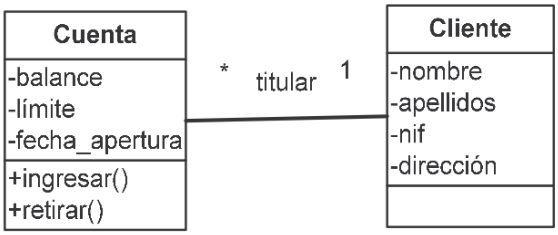
equivale a



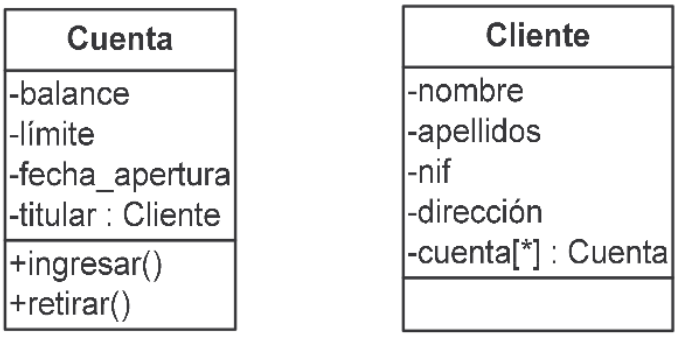
```
class Cuenta{
	private Dinero balance;

	public void ingresar (Dinero cantidad)
	{
		balance += cantidad;
	}
	public void retirar (Dinero cantidad)
	{
		balance -= cantidad;
	}
	public Dinero getSaldo ()
	{
		return balance;
	}
}
```

Se ha supuesto que Dinero es un tipo de dato con el que se pueden hacer operaciones aritméticas y por tanto, se ha añadido un método adicional que nos permite comprobar el saldo de una cuenta.



Este esquema viene a ser lo mismo que el siguiente con la salvedad de que el enlace bidireccional lo tenemos que mantener nosotros



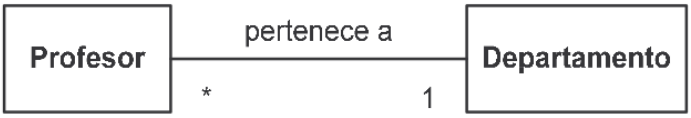
Un cliente puede tener varias cuentas, por lo que en la clase cliente hemos de mantener un conjunto de cuentas (un vector en este caso).

Hay que tener en cuenta:

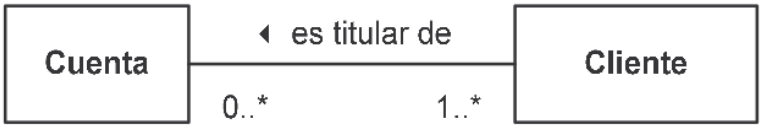
- La multiplicidad de una relación determina cuántos objetos de cada tipo intervienen en la relación:
- Para especificar la multiplicidad de una relación hay que indicar la multiplicidad mínima y la multiplicidad máxima (mínima..máxima)
- Cuando la multiplicidad mínima es 0, la relación es opcional.
- Una multiplicidad mínima mayor o igual que 1 establece una relación obligatoria.

Ejemplo:

Todo departamento tiene un director. Un profesor puede dirigir un departamento.



Todo profesor pertenece a un departamento. A un departamento pueden pertenecer varios profesores.

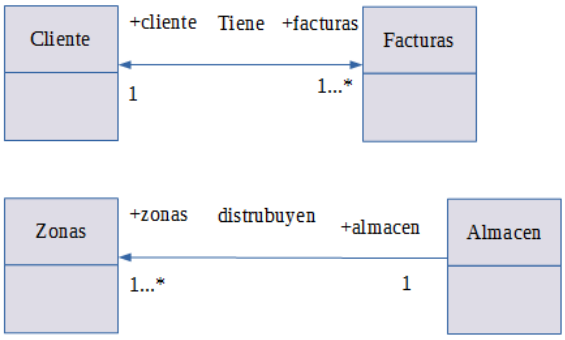


Relación opcional
Un cliente puede o no ser titular de una cuenta

Relación obligatoria
Una cuenta ha de tener un titular como mínimo

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

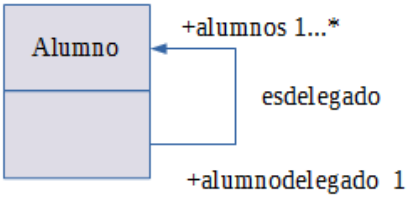
Ejemplo de relaciones de asociación



a) En la primera asociación “Tiene” muestra que un cliente tiene muchas facturas , y la factura es de un cliente, como es bidireccional ambas clases conocen su existencia, ambas clases son navegables.
b) En la asociación “distribuyen”, un almacén distribuye artículos en varias zonas. La asociación es unidireccional , sólo la clase origen Almacén conoce la existencia de la clase destino Zonas.

Asociación reflexiva o involutivas

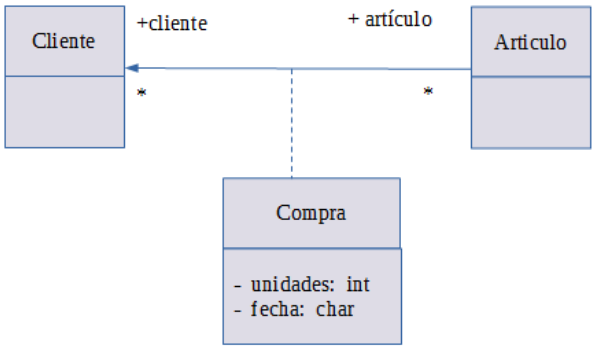
Se da cuando una clase se asocia consigo misma. Estas asociaciones unen entre si instancias de una misma clase



Un alumno (delegado) es delegado de muchos alumnos

Clase Asociación

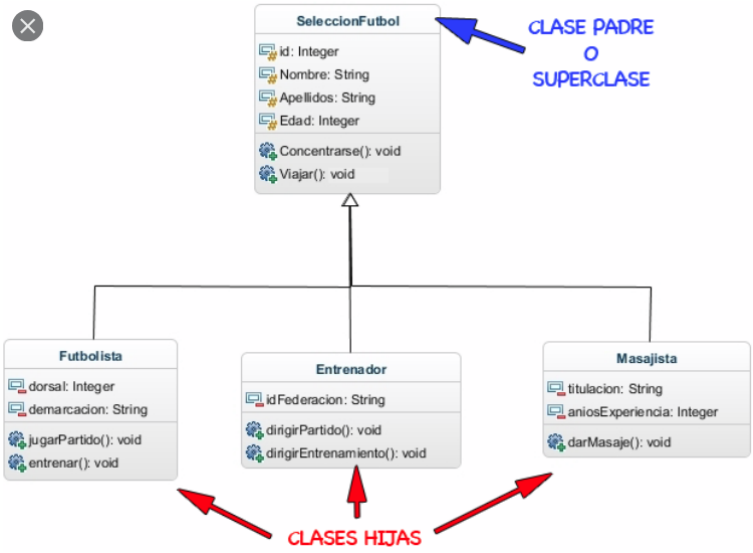
Una asociación entre dos clases puede llevar información necesaria para esa asociación , a esto se le llama clase asociación, es como una relación N:M con atributos del modelo Entidad/Relación. En este caso, esta clase asociación recibe el estatus de clase y sus instancias son elementos de la asociación , estas clases pueden estar dotadas de atributos y operaciones y estar vinculadas a otras clases a través de asociaciones.

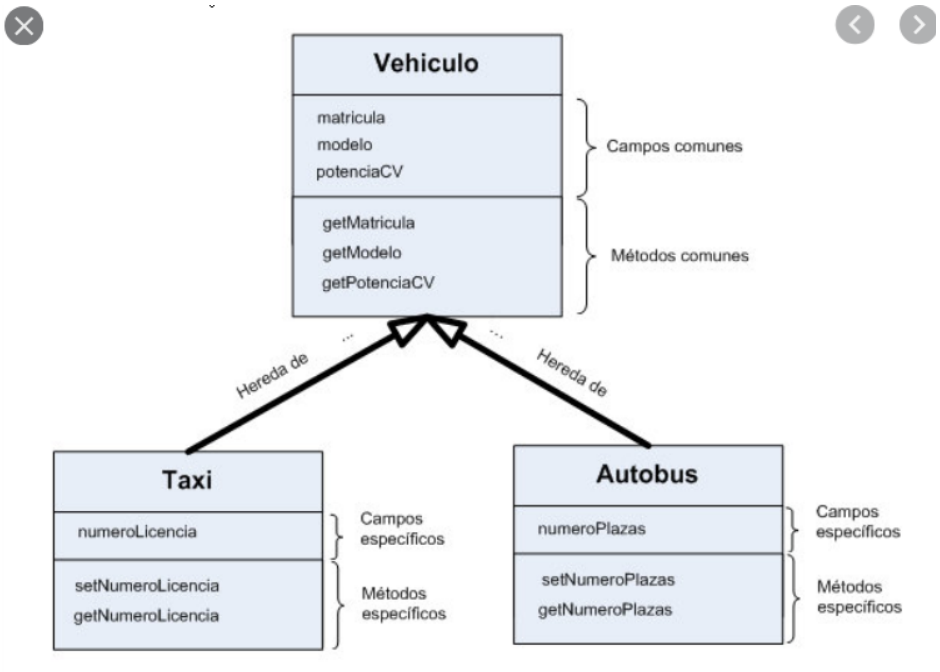


La clase asociación Compra , un cliente compra muchos artículos, un artículo es comprado por muchos clientes, y de la compra se necesita saber la fecha de compra y las unidades compradas.

HERENCIA (GENERALIZACIÓN Y ESPECIALIZACIÓN)

La herencia es una abstracción importante para compartir similitudes entre clases, donde todos los atributos y operaciones comunes a varias clases se pueden compartir por medio de la superclase, una clase más general. Las clases más refinadas se conocen como las subclases . La generalización define una relación entre una clase más generalizada, y una o más versiones refinadas de ella. La generalización indica que una clase (subclase) hereda los atributos y métodos de otra (superclase). La superclase generaliza a sus subclases, y las subclases especializan a la superclase.





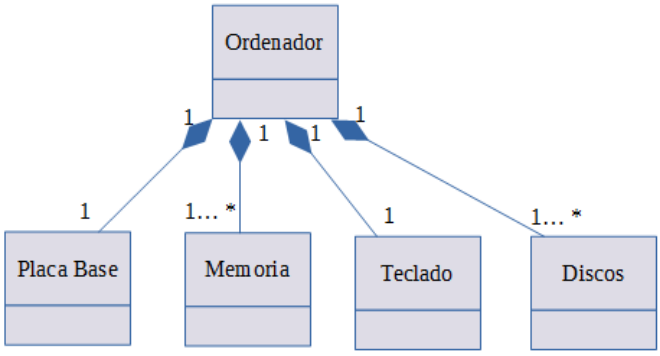
COMPOSICIÓN

Un objeto puede estar compuesto por otros objetos , en estos casos nos encontramos ante una asociación entre objetos llamada “Asociación de composición”. Esta asocia un objeto complejo con los objetos que lo constituyen, es decir, sus componentes. Existen dos formas de composición , fuerte o composición y débil o agregación.

Composición fuerte o agregación

En la composición fuerte los componentes constituyen una parte del objeto compuesto y estos no pueden ser compartidos por varios objetos compuestos. Por tanto , la cardinalidad máxima, a nivel del objeto compuesto, es uno. La eliminación del objeto compuesto conlleva la eliminación de los componentes.

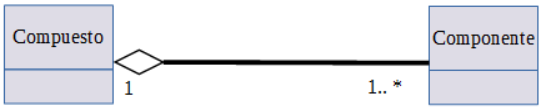
Ejemplo: Asociación de composición entre un ordenador y sus componentes.



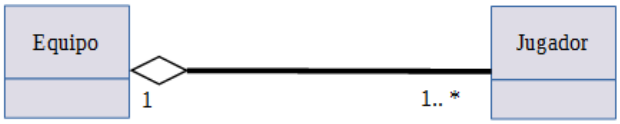
El ordenador se compone de una placa base, uno o varios módulos de memoria, un teclado y uno o varios discos.

Agregación o composición débil.

Los componentes pueden ser compartidos por varios compuesto y la destrucción del compuesto no implica la destrucción de los componentes.



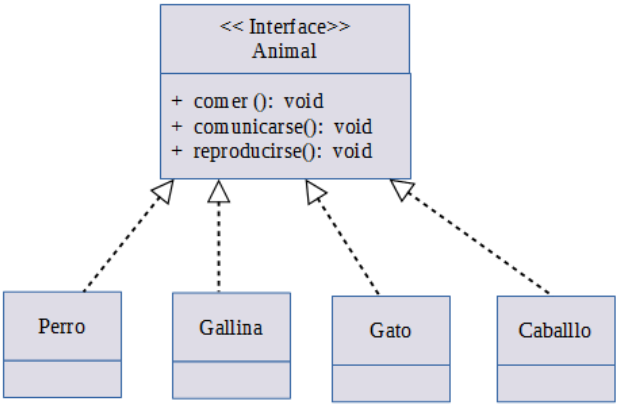
Ejemplo de asociación de agregación entre una clase Equipo , y la clase Jugador . Un equipo está compuesto por jugadores , sin embargo, el jugador puede jugar también en otros equipos. Si desaparece el equipo , el jugador no desaparece



REALIZACIÓN

Una relación de realización es la relación de herencia existente entre una clase interfaz y la subclase que implementa esa interfaz.

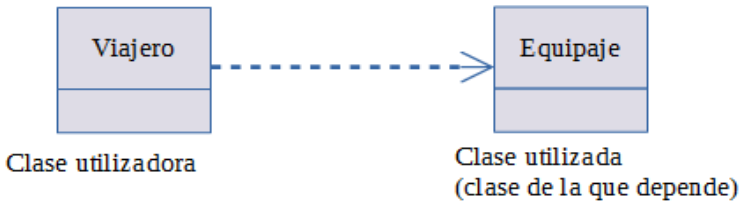
Nota: Una interfaz es una clase totalmente abstracta , es decir , no tiene atributos y todos sus métodos son abstractos y públicos , sin .



En la figura tenemos una asociación de realización entre una clase interfaz Animal y las clases Perro, Gallina, Gato, Caballo. Se considera que cualquier animal come, se comunica y se reproduce, sin embargo, cada animal lo hace de una forma distinta. Cada subclase implementará los métodos de la interfaz.

DEPENDENCIA

Es una relación que se establece entre dos clases cuando una clase usa a la otra , es decir , que la necesita para su cometido, las instancias de la clase se crean y se emplean cuando se necesitan.



Con la dependencia mostramos que un cambio en la clase utilizada puede afectar al funcionamiento de la clase utilizadora, pero no al contrario

En la relación entre un Viajero y su Equipaje, el viajero necesita su equipaje para viajar, la clase Viajero depende de la clase Equipaje porque la necesita.

4.1.3.- Estereotipos

Son mecanismo de extensibilidad que más se utiliza dentro del modelado UML, este mecanismo va a permitir definir nuevos elementos de modelado UML basándose en uno existente. Podemos definir la semántica (significado, interpretación y sentido) del estereotipo. Puede ser aplicado a cualquier elemento de modelado como clases, paquetes, relaciones de herencia, o cualquier otro tipo de relación.

Cada estereotipo puede definir un conjunto de valores etiquetados y restricciones que se aplican al elemento estereotipado. El nombre del estereotipo generalmente se indica "« » ", por ejemplo **«interface »** , aunque tambien se les puede asociar un icono.

Existen muchas formas de representar un estereotipo.

NOTAS:

Otros mecanismos de estensibilidad de UML son:

- **Las restricciones**: amplían la semantica de un elemento al permitir añadir nuevas reglas, se representa como una cadena de texto entre llamves { } que especifica la condición o regla sobre el elemento.
- **Los valores etiquetados**: proporcionan una forma de ampliar la especificación de un elemento al permitirnos añadir nueva información en él. Estos valores van entre llaves con la sintaxis {etiqueta=valor. . . }
- **Un perfil UML** es una colección de estereotipos, valores etiquetados y restricciones. Los perfiles se utlizan para personalizar los diagramas UML para una finalizad específica.

Es conveniente conocer e identificar los estereotipos que proporcionan las herramientas de modelado , algunos de ellos son los siguientes:

- **Estereotipos para los diagramas de clases**: Enumeración, Interfaces, DataType, Signal, Exception, entre otros.
- **Estereotipos para los diagramas de comportamiento**: *Entity, Control, y Boundary*.

Boundary

Representa una clase mediadora entre el sistema y su entorno. Son clases que hacen de interface entre el sistema y los usuarios, entre el sistema con otros sistemas , y el sistema con otros dispositivos externos. Se usan para modelar la interacción entre el sistema y los actores , esta interacción involucra recibir y presentar informaciones y peticiones desde los usuarios y sistemas externos.

A los objetos de esta clase se les llama ***instancia de clases límite o frontera***. Estas clases representarán por ejemplo las ventanas , formularios, impresoras o dispositivos de nuestro sistema.

Control

Estas clases son controladoras, sus instancias coordinan el comportamiento del sistema correspondientea uno o más casos de uso. Los objetos control modelan la funcionalidad del sistema, representan la coordinación , secuencia, gestión de transacciones y control de otros objetos.

Se usan para representar cálculos y derivaciones complejas, como la lógica del negocio que no se puede relacionar con ninguna entidad.

La dinámica del sistema se modela en una clase controladora, que se encarga de delegar trabajo a otras clases. Por ejemplo, una clase que modele la gestión de artículos de un almacen.

Entity

Estas clases guardan información sobre el estado interno del sistema , a corto y largo plazo, corresponde al dominio del problema. Expresa la estructura lógica de datos del sistema y están íntimamente relacionadas con el modelo de datos.

Están manipuladas por la clase de control y aceptan información de clases límite.

Normalmente son clases persistentes (por ejemplo asociadas a tablas de bases de datos relacionales), ejemplo asociadas a tablas de bases de datos relacionales), ejemplo de ellas pueden ser la clase artículos o la clase clientes.