Diagrama de Clases

**Introducción**

Sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de consentimiento.

Un diagrama de clases está compuesto por los siguientes elementos:

* [Clase](http://users.dcc.uchile.cl/%7Epsalinas/uml/modelo.html#clase): métodos y visibilidad.
* [Relaciones](http://users.dcc.uchile.cl/%7Epsalinas/uml/modelo.html#relacion): Herencia, Composición, Agregación, y Uso.

**Elementos**

* **Clase**

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

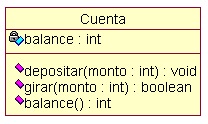
En UML, una clase es representada por un rectángulo que posee tres divisiones:



En donde:

* + **Superior**: Contiene el nombre de la Clase
  + **Intermedio**: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
  + **Inferior**: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

El diseño asociado es:



Atributos y Métodos:

* + **Atributos:**

Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

* + **Métodos:**

Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

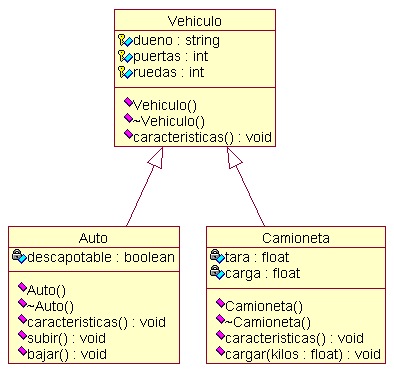
* **Relaciones entre Clases:**

Ahora ya definido el concepto de Clase, es necesario explicar cómo se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes).

Antes es necesario explicar el concepto de cardinalidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

* + **uno o muchos**: 1..\* (1..n)
  + **0 o muchos**: 0..\* (0..n)
  + **número fijo**: m (m denota el número).
  + **Herencia (Especialización/Generalización)**: http://users.dcc.uchile.cl/%7Epsalinas/uml/img/modelo/herencia1.jpg

Indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected), ejemplo:



En la figura se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (Precio, VelMax, etc) además posee algo particular que es Descapotable, en cambio Camión también hereda las características de Vehiculo (Precio, VelMax, etc) pero posee como particularidad propia Acoplado, Tara y Carga.

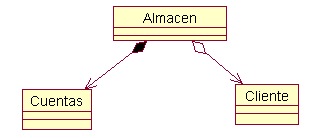
Cabe destacar que fuera de este entorno, lo único "visible" es el método Caracteristicas aplicable a instancias de Vehículo, Auto y Camión, pues tiene definición pública, en cambio atributos como Descapotable no son visibles por ser privados.

* + **Agregación**: http://users.dcc.uchile.cl/%7Epsalinas/uml/img/modelo/agregacion1.jpg

Para modelar objetos complejos, n bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

* + - **Por Valor**: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comunmente llamada **Composición** (el Objeto base se contruye a partir del objeto incluido, es decir, es "parte/todo").
    - **Por Referencia**: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comunmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).

Un Ejemplo es el siguiente:



En donde se destaca que:

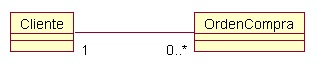
* + - Un Almacen posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).
    - Cuando se destruye el Objeto Almacen también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
    - La composición (por Valor) se destaca por un rombo relleno.
    - La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto refereniado. Cuando no existe este tipo de particularidad la flecha se elimina.

* + **Asociación**: http://users.dcc.uchile.cl/%7Epsalinas/uml/img/modelo/asociacion1.jpg

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo:



Un cliente puede tener asociadas muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

* + **Dependencia o Instanciación (uso)**: http://users.dcc.uchile.cl/%7Epsalinas/uml/img/modelo/dependencia1.jpg

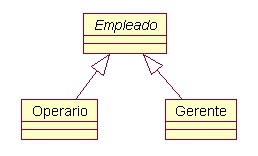
Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada.

El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación grafica que instancia una ventana (la creación del Objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicacion):

http://users.dcc.uchile.cl/%7Epsalinas/uml/img/modelo/edependencia.jpg

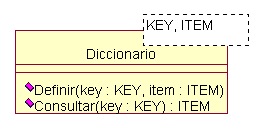
Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

* **Casos Particulares**:
  + **Clase Abstracta**:



Una clase abstracta se denota con el nombre de la clase y de los métodos con letra "itálica". 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 implementan los métodos abstractos definidos.

* + **Clase parametrizada**:



Una clase parametrizada se denota con un subcuadro en el extremo superior de la clase, en donde se especifican los parámetros que deben ser pasados a la clase para que esta pueda ser instanciada. El ejemplo más típico es el caso de un Diccionario en donde una llave o palabra tiene asociado un significado, pero en este caso las llaves y elementos pueden ser genéricos. La genericidad puede venir dada de un Template (como en el caso de C++) o bien de alguna estructura predefinida (especialización a través de clases).

En el ejemplo no se especificaron los atributos del Diccionario, pues ellos dependerán exclusivamente de la implementación que se le quiera dar.

**Ejemplo**:

Supongamos que tenemos tenemos un el caso del Diccionario implementado mediante un árbol binario, en donde cada nodo posee:

* key: Variable por la cual se realiza la búsqueda, puede ser generica.
* item: Contenido a almacenar en el diccionario asociado a "key", cuyo tipo también puede ser genérico.

Para este caso particular hemos definido un Diccionario para almacenar String y Personas, las cuales pueden funcionar como llaves o como item, solo se mostrarán las relaciones para la implementación del Diccionario:

