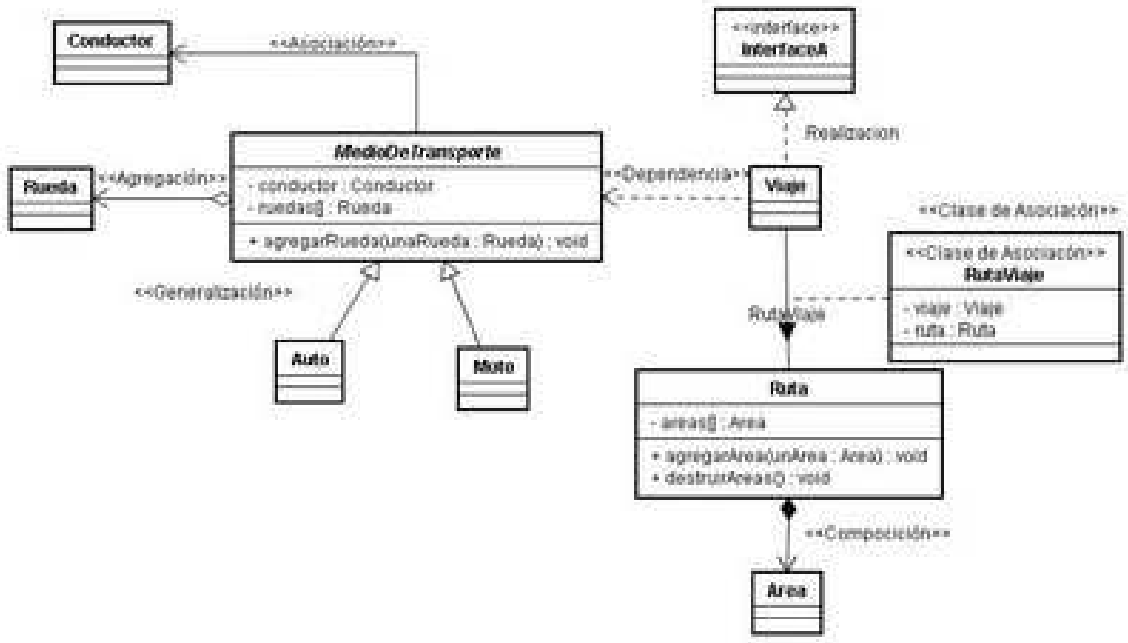
Interfaces

A las clases abstractas puras, es decir, a las clases que no contienen ninguna implementación, se les llama interfaces. En UML una interfaz es una colección de operaciones que sirven para especificar los servicios de una clase o un componente. Una interfaz sólo contiene las cabeceras de las operaciones, no su implementación. (Una interfaz de UML se corresponde con una clase virtual pura de C++ y con una "interface" de Java). Gráficamente una interfaz se puede representar de forma expandida como una clase estereotipada con la etiqueta <<interface>> o, en su forma abreviada, con una figura en forma de piruleta. En los diagramas de clases se suele utilizar la forma expandida para representar las interfaces. La forma abreviada generalmente se usa en los diagramas de componentes. Hay dos relaciones que pueden existir entre una clase y una interfaz: la dependencia y la realización. La dependencia entre una clase y una interfaz tiene el mismo significado y representación que entre dos clases, indica que la clase usa la interfaz. Para que una interfaz se pueda usar hace falta que otra clase implemente las operaciones que la interfaz especifica. A esta relación entre la interfaz y la clase que la implementa se le llama realización. La realización indica que la clase implementa todas las operaciones de la interfaz. Gráficamente la realización se representa como una generalización con la línea discontinua.

Relaciones, Composición, Agregación, Asociación, Dependencia, Generalización, Realización



Asociación:

Es generalmente, una relación estructural entre clases, es decir, que en el ejemplo, existe un atributo de la clase medio de transportes, que es del tipo Conductor. La navegabilidad nos muestra donde está ubicado el atributo. Es decir cuál es ***la clase que tiene contiene el atributo si ésta no lo mostrase***.

Agregación:

Es una relación que se derivó de la asociación, por ser igualmente estructural, es decir que contiene un atributo, que en todos los casos, será una colección, es decir ***un array, vector***, etc, y además de ello la clase que contiene la colección debe tener un método que agregue los elementos a la colección. ***También se puede leer como que un medio de transporte tiene varias ruedas***.

Nos está diciendo que los objetos rueda forman parte del objeto medio de transporte. Pero, su ciclo de vida no está atado al del objeto medio de transporte. ***Es decir si el automóvil se destruye las ruedas pueden seguir existiendo independientemente***.

Composición:

Al ***igual que en la agregación***, es una relación estructural ***pero se le suma***, que tiene un método de destrucción de los objetos. Y a ***diferencia*** de la asociación, el ciclo de vida del objeto área ***está relacionado con el del objeto ruta***. Es decir que si la ruta de viaje se levanta, las áreas que surgían a partir de ella desaparecen. También se puede leer como que una ruta tiene varias áreas de cobertura.

Mucho se ha discutido a cerca de las agregaciones y las composiciones, el debate es casi tan caliente como el de los include y extends de los casos de uso. Ya que algunos sostienen que los lenguajes orientados a objetos, tienen garbage collector, por lo que no necesitan métodos de destrucción de los objetos (relacionados a los ciclos de vida en la composición). Y que la programación es la misma para las composiciones y las agregaciones, y que la diferencia es meramente conceptual entre una y otras. Es más, existen varias interpretaciones al respecto.

Clase de Asociación:

Es una Clase que surge de una multiplicidad de muchos a muchos, y fue incorporada en UML para dar soporte a este caso. Se sacan los atributos de las clases involucradas y se los incorpora a una clase a parte. Al igual que las anteriores hace referencia a una relación estructural. En el ejemplo son los objetos viaje y ruta

Realización: 

Es una relación de contrato con otra clase. Se la utiliza para implementar una interfaz. En lenguajes como java o php utilizamos la palabra reservada “implements”

```
public class Viaje implements InterfaceA{…}
```

Generalmente cuando no estamos seguros si “algo” es una interfaz o una clase abstracta, porque dibujaron los tag que hacen referencias a las interfaces, debemos ver la relación para saber.

Generalización o Herencia: 

Es una relación de herencia. ***Se puede decir que es un relación “es un tipo de”***. En nuestro ejemplo: “un auto es un tipo de Medio de transporte”. Es entre una clase hija y su clase madre. En la codificación podemos encontrar la palabra “extends” que hace referencia a esta relación. Además podemos encontrar palabras claves tales como “this” y “super” o "self" y “parent”. Para darnos cuenta que existe una relación de este tipo involucrada.

```
public class Auto extends MedioDeTransporte{…}
```

Dependencia: 

Es una relación de uso, es decir que una clase utiliza a otra. Y si esta última se altera, la anterior se puede ver afectada.

En código se suelen traducir principalmente como las clases donde se hace la instanciación de un objeto. En nuestro ejemplo la clase viaje realiza los “new” de los distintos objetos. En este momento puede que te preguntes como puede hacer un new de una clase abstracta. No realiza los new de la clase abstracta, si no de sus hijas. Sería algo así como

```
MedioDeTransporte medio =new Auto();
```

También se sostiene que este tipo de relación hace referencias, a los parámetros que se pasan en un método, bajo este concepto, en java, podría ser algo así como:

```
public void crearViaje(MedioDeTransporte medio){}
```

Por ultimo también se sostiene que podemos codificar esta relación realizando un “return” del tipo de dato en algún método.