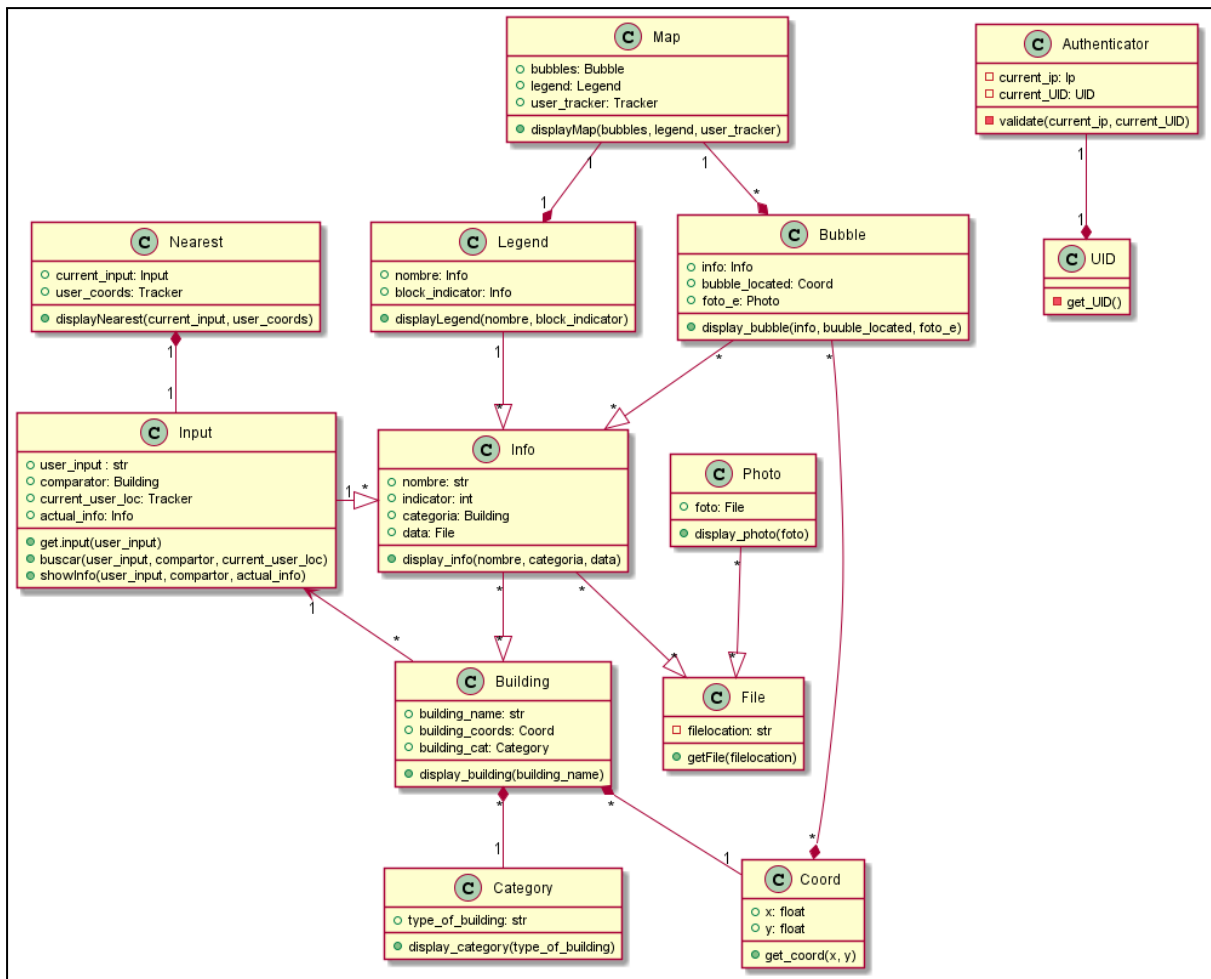


A partir del siguiente diagrama analiza las relaciones entre clases, multiplicidad y atributos:



(Diagrama UML)

Como se puede ver en el diagrama UML de arriba, tenemos varias clases relacionadas entre sí de la siguiente forma:

1. La clase *Authenticator* guarda una **relación de Composición** con la clase *UID*, pues si esta última clase es eliminada, también se debería eliminar la clase *Authenticator*, pues esta no podría existir sin *UID*.
2. Por su parte, la clase *Map* también guarda una **relación de Composición** con las clases *Legend* y *Bubble*, de forma que si una de estas es eliminada, la clase *Map* se quedaría incompleta pues requiere de ambas para poder funcionar correctamente.
3. De forma inversa, las clases *Coord* y *Category* guardan también una **relación de Composición** con la clase *Building*, de tal forma que si la clase *Building* desaparece también lo deberían hacer *Coord* y *Category*.





4. De hecho, la clase *Bubble* guarda una **relación de Composición** con la clase *Coord*, de tal forma que *Bubble* depende de *Coord* para su correcto funcionamiento. De forma similar que como ocurre con las clases *Input* y *Nearest*, que guardan el mismo tipo de relación y además *Input* es dependiente de *Nearest*.
5. Por otro lado, en el diagrama *UML* presentado en la imagen existen varias clases derivadas (o subclases) de unas clases padres (o superclases) que estarían distribuidas de la siguiente forma basándonos en su **relación de Herencia**:
 - a. Las clases *Input*, *Legend* y *Bubble* son clases derivadas de la clase *Info*.
 - b. Por su parte, *Info* es una clase derivada tanto de *Building* como de *File*.
 - c. De forma similar, la clase *Photo* también es una clase derivada de *File*.
6. Finalmente, *Building* guarda una **relación simple** con *Input*.

Explicado los tipos de relaciones mostrados, sería hora de tratar la multiplicidad en el diagrama *UML*, el cual sería el siguiente:

1. Entre *Authenticator* y *UID* hay una **relación de 1 a 1**, pues por cada *UID* habrá tan solo un *Authenticator*.
2. Algo similar ocurre entre *Map* y *Legend* y entre *Input* y *Nearest*. De tal forma que por cada *Map* habrá un *Legend*, el cual solo podrá estar asociado a un único *Map*. Mientras que un *Input* tan solo puede asociarse con un único *Nearest* y viceversa.
3. Entre *Map* y *Bubble* hay una relación de **1 a varios (*)**, pues es posible que existan varias *Bubbles* dentro de un *Map*, y que dichas *Bubbles* solo puedan estar asociadas a un único *Map*.
4. Algo similar vemos con *Coord* y *Building*, donde pueden haber varias *Building* para una única *Coord*. O como ocurre con *Category* y *Building*, donde pueden haber varias *Building* para una misma *Category*. Y también sucede lo mismo con *Input* y *Building*, donde es posible tener un único *Input* que afecte a varios *Building*.
5. De igual forma, es posible que un mismo *Legend* o un único *Input* contengan varias *Info*, explicando así su multiplicidad **1 a varios (*)**.
6. Finalmente, las relaciones de **varios (*) a varios (*)** se dan sobre un par de relaciones, como en el caso de *Bubble* y *Coord*. Esto se debe a que pueden existir varias *Coord* para varias *Bubble*.
7. Algo similar ocurre entre *File* y *Photo* y entre *File* y *Info* también hay una relación de **varios (*) a varios (*)**, donde pueden haber varios *Files* que sean *Photos* o *Infos*.
8. Lo anterior también aplica para la relación entre *Info* y *Building* y entre *Bubble* e *Info*.

Finalmente, en el diagrama mostrado existen varios atributos y métodos que merecen ser explicados en conjunto. Como se puede ver, antes de cada atributo y/o método hay una forma

geométrica, la cual puede estar llena o vacía. Dependiendo de esta forma, el atributo/método se comportará de una forma u otra. Para entender esto, la siguiente lista nos puede ayudar:

1.  *Círculo Verde Vacío* = **Privado**: Estos atributos tan solo podrán ser usados dentro de la misma clase. Por ejemplo, el atributo `current_input` tan solo podrá ser modificado explícitamente en el interior de clase `Nearest`.
2.  *Círculo Verde* = **Público**: Estos métodos podrán ser utilizados por cualquiera clase dentro del mismo proyecto.
3.  *Cuadrado Naranja Vacío* = **Protegido**: Estos atributos podrán ser utilizados por las clases derivadas de una superclases y por la propia superclase.
4.  *Cuadrado Naranja* = **Paquete/Defecto**: Estos métodos podrán ser vistos por el resto de métodos de clases que estén dentro de un mismo paquete, es decir, que estén relacionadas.