UML and relations

Classes and class diagrams

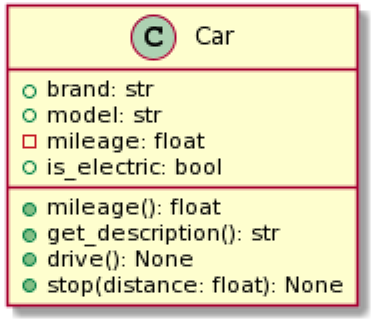
- Classes and objects have
  - attributes
  - behaviours
- We can easily represent classes using a diagram, called a *class diagram*
- Note: in this course, the class diagrams will usually only show the public interface, and not the private members
- Encapsulation and abstraction
- UML is just a common language: standards vary by team / project

Required items in a class diagram

The class diagram shows:

- the name of the class
- the attributes and their types
- the methods, with:
    - the parameters and their types
    - the return value and its type

What about the `Car` class?



Looks complicated!

Comments

- The `@property` decorator can be confusing: it "looks like" an attribute, but works through methods
- It is generally located in the methods section of the class diagram (see above)
- The convention in Python is to use `_name` for a private variable called `name`.
- In UML, you use a `-` for private attributes, and `+` for public attributes.

Objects and relationships

- Objects do not exist by themselves
- Like their real-life reprensentations, they exist in a context
- Objects interact with each other
- Objects are linked by relationships

Examples

- Between two persons A and B:
    - A *is married* to B
    - A *is the father* of B
    - A *is the boss* of B
- Between a hand and a finger:
    - A hand *has* five fingers
    - A finger *belongs to* a hand

Relations in UML diagrams

They are simply represented by a line connecting the two classes. Sometimes, the *nature* of the relationship is expressed in the diagram. There are different types of relationships:

- composition
- aggregation
- extension (or inheritance)

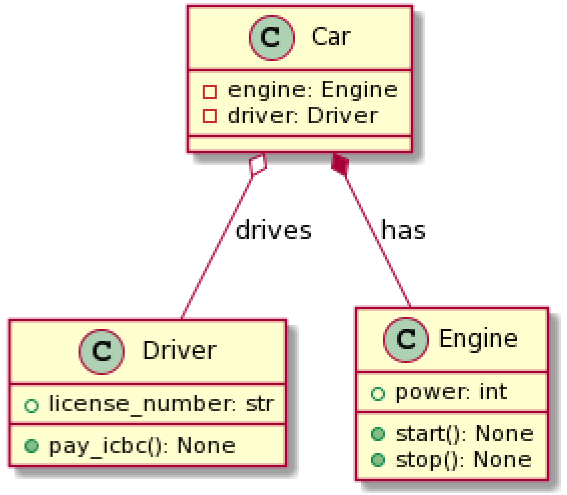
Each relationship has a different "line" representation.

Aggregation vs composition

- Aggregation is when two objects are related, but can "exist" independently of each other
- Composition is when one object cannot exist without the other
- A car "system" is made of a car and a driver
- A car is made of an engine and a body

- If the car is destroyed, the driver still exists
- If the car is destroyed, the engine does not exist anymore

Aggregation vs Composition: UML



> Notice the difference between aggregation and composition in the class diagram.