Object Orientated
Class Relationships

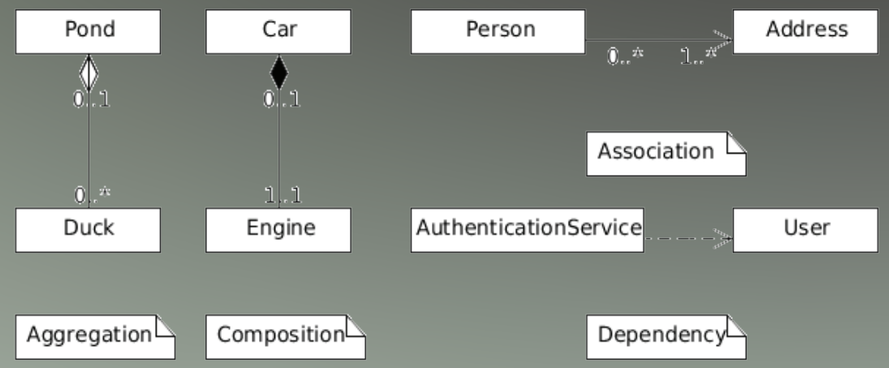
OOP (object oriented programming)
A paradigm centered around objects and data rather than actions and logic. It is imperitive in Object orientated programming to identify the objects along with there relationships. The essential relationships between objects include: association, generalisation, specialisation, aggregation, dependency and composition.

Generalisation & Inheritance
Inheritance allows new objects to take on the properties/methods of existing objects. A class that is used as the basis for inheritance is called a superclass, parent class or base class. A class that inherits from a superclass is called a subclass, child class or derived class.

Generalisation is the process of extracting shared characteristics from two or more classes and combining them into a generalised superclass. Shared characteristics can be attributes, associations, or methods.

Realisation
A UML term used to denote a semantic relationship between two entities, in which one entity or type gurantees to fufill (implements or executes) the behavior that the other entity or type specifies.

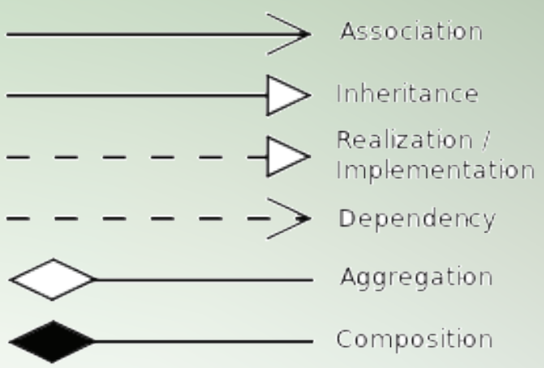
Dependency
A relationship between two or more objects in which an object is dependant on another object or set of objects for its implementation. If one of these objects change, the other object(s) can be impacted. Dependency is commonly depicted using dashed arrows and used in a UML diagram as shown below:



Aggregation | Composition
Subsets of association meaning they are specific cases of association. In both, one class "owns" an object of another class but there is a subtle difference:

Aggregation implies the relationship where a child can exist independently and without its parents class. Example: Class (parent) and student (child). Delete the class and the students still exist.

Composition however, implies a relationship where the child cannot exist without it's parent class, meaning that if the parent or base class is deleted , the attached child or sub classess also get deleted. Example: House (parent) and room (child). Rooms don't exist separate to a House.



Loose & Tight Coupling
Coupling refers to the degree to which the different modules/classes depend on each other.

Loose Coupling is an approach to interconnecting the components in a system or network, with the intent of ensuring these components rely on each other to the least extent practicable.

Tight coupling is the complete opposite to loose coupling, where groups of classes are highly dependent on one another. This occurs when a class assumes too many responsibilities, or when one concern is spread over many classes rather than its own personal class.

Low & High Cohesion
Coupling is usually contrasted with cohesion. Loose coupling often correlates with high cohesion, and vice versa. In OOP, cohesion refers to the degree to which the elements inside a module belong together. Thus high cohesion can be seen as a measurement of how strongly related each piece of functionality expressed by the source code of a software module is (which elements of a certain module belong together).

In contrast to how high cohesion modules are preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability,. Low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand.