Unified Modeling Language: Class Diagrams

Tags

> **Topics covered today:**

- Class Diagrams
- Sequence Diagrams
- Communication Diagrams

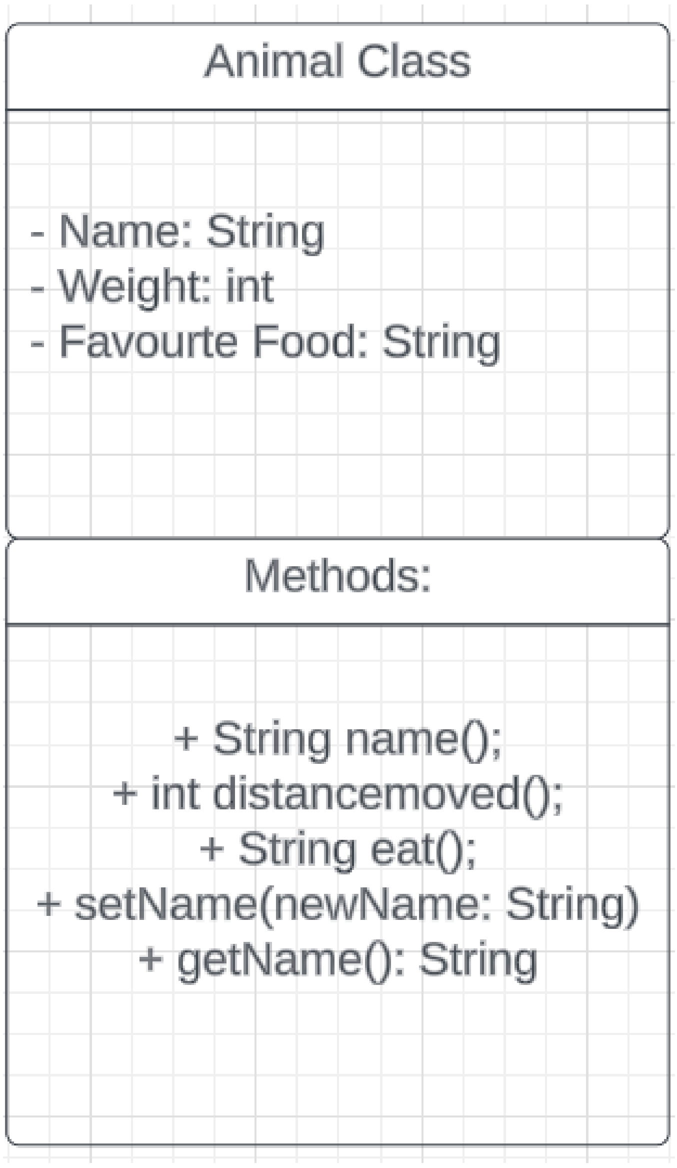
A unified Modeling Language was created to implement in visual representations a complex architectural design of softwares.

- The Agile and Waterfall methods in their graphical form are examples of UMLs
- The Agile development process is a adaptive planned process
- A UML is a graphical way of describing software systems

- a UML sketch is an overall look at the system you are trying to create
- Whereas a UML Blueprint is very detailed

Class Diagrams:

- Classes describe the type of objects that the system is going to use
- A class is made up of states / attributes on the top and Behaviours / Methods
- e.g. The animal class diagram below:



- The '-' and '+' represent visibility i.e. is the field private / protected or public.
- A protected visibility is specified by a #
- A ~ next to a attribute or method specifies package which means as long as the method is in the same package the class can be used by any other class
- Notice how the attributes are all private i.e. only visible to that class becuase class attributes should be private so that they cannot be altered by other fields. A generic abstract class should have private fields.

- The methods / behaviours should be protected i.e. only classes that inherit or extend the class can access the methods of the class.
- The nouns in the task description can be classes. But nouns can also be properties in the class:
-

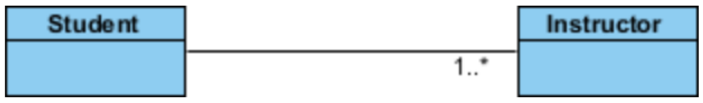
Multiplicity:

- Multiplicity is the number of instances one class relates to ONE instance of another class
- For example a animal class might relate once to a dog class, for each dog class you have identified an animal
- for Favourite food, in the animal class we could have it as a list of favourite foods this is declared using Food: [1 .... 10]
- If the number is unknown then this is represented by a '*' so you can have 0....*
- **Static Attributes** are those attributes that can be shared between all of the class objects that are created from that class

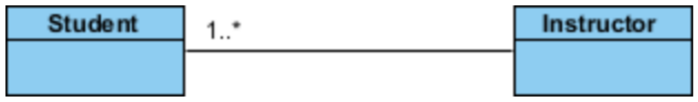
Association

- If two classes in a model need to communicate with each other, there must be a link between them, this can be represented by a line between them.
- The multiplicity is shown on the arrow

A single student can associate with multiple teachers:



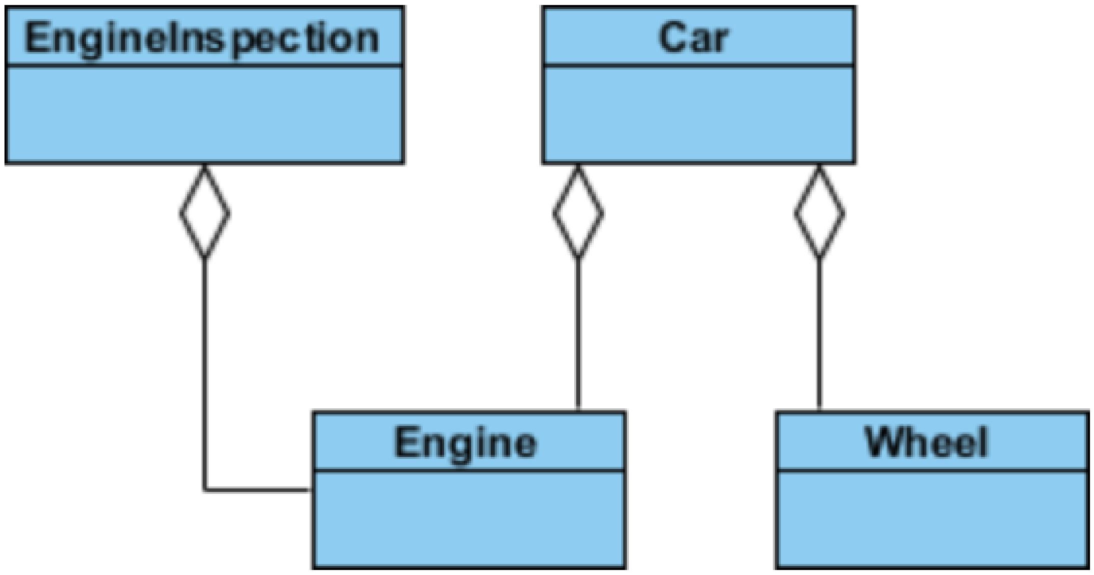
The example indicates that every Instructor has one or more Students:



Aggregation and Composition

- Aggregation and compositions are subsets of association, they each describe a parent child relationship, i.e. one class directly owns the attributes the other class is using. Another term you might see is the parents class being refered to as a “Whole” and the children class being refered to as “parts”
- In aggregation however, the child class can exist independent of the parents class. For example, A classroom (the parent class) can have attributes that describe students, it is the parents class to the student class. However, if you

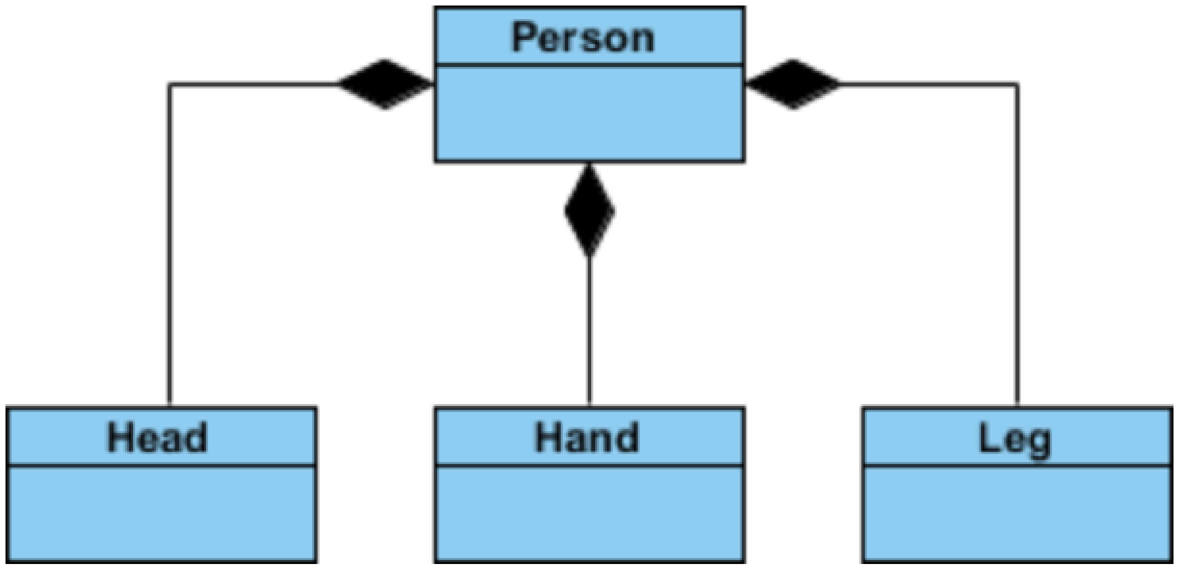
delete the classroom class the students will still exist, it has other attributes for example, height, student ID etc, these attributes are independent from the classroom class.



- Another example you can look at is a zoo, imagine you have a class that represents a group or crows called a "Murder" a single crow class can exist as part of the Murder class but if you deleted the murder class i.e. if you the zoo did no longer have a group of crows, each individual crow would still exist as its own entity.
- A aggregate relationships such as theese are represented by a open diamond arrow

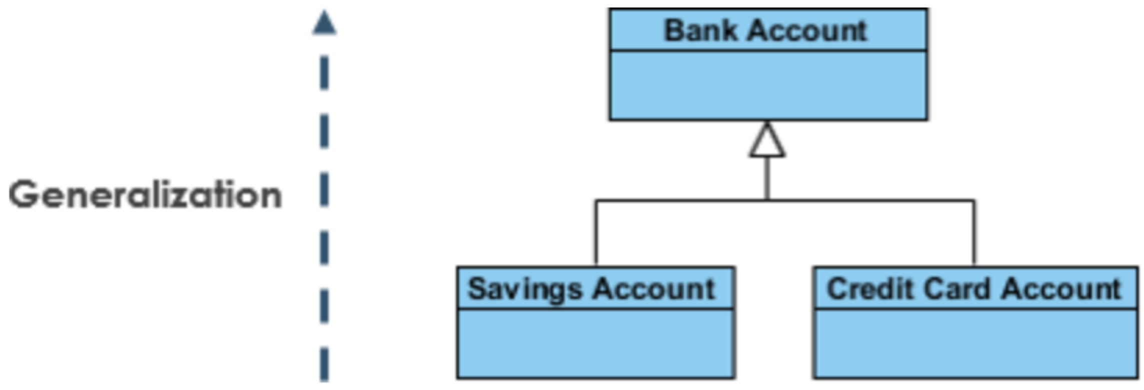
Composition

- A composition is where a child class cannot exist without a parents class e.g. House (parent class) has a child called Room (child class), a room cannot exist without a house



Generalisation / Inheritance

- Generalisation is the mechanism of combining two or more classes that share attributes / methods into a single entity
- Generalisation is represented by a open arrow as shown below:
- For example a savings account and credit car accounts have lots of similar information, such as a persons name, their account info and so on. So these can be generalised to a single class called bank account nt
- You are still keeping the savings account and credit card account as spereat classes however, because they also contain information that is independent form each other like interest will be on the credit card account but not the savings account



- Another example can be: Squares, Rectangle and Circle all belong to parent class called shapes.
- The Common attributes that can describe them is e.g. number of sides
- WIth Inheritance another key word is also useful: **Abstraction**
    - Abstraction is the parent class such as the shape class that all shapes belong to, it is an abstract class becuase the class itself is never instantiated.
    - You can represent an abstract class by either puttong the class name in italics or in “<<>>” e.g.: <<Shapes>>