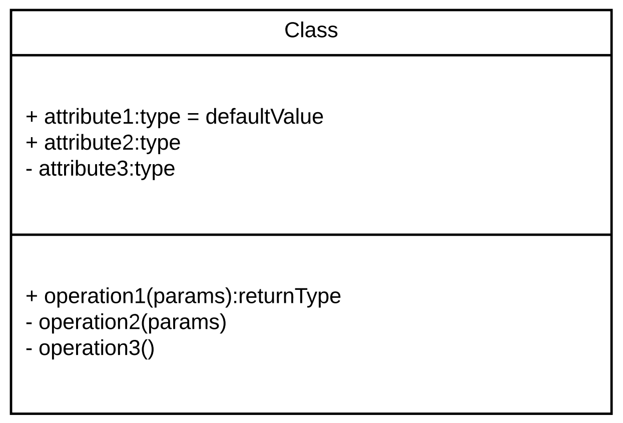
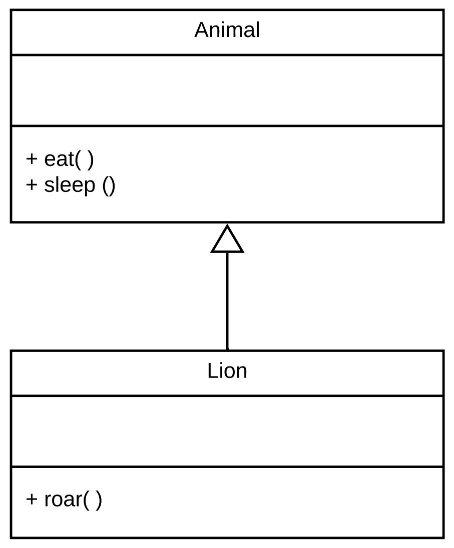


class names are typically capitalized while instances are lower case.



When a class uses an interface we say that the class implments the interface.

Aggregation: You can have a car without an engine, and you can have an engine without a car. Granted neither will do much but you can still do that.

Composition: You have a bedroom and a house. If you tear the house down the bedroom goes too. They are tightly cooupled and not mutially exclusive.

Using concepts like aggregation and delegation OO style development allows you to abstract the complexities of a task out hiding the difficulty of the task behind a simple interface.

Class

A blueprint for objects. Like a floor plan for a subdivisoin. You have 3 types of homes. Each home is an *instance* of that class. Each home has it's own *attributes* or properties in classical nomenclature. A class mainly consists of methods (functions that belong to a class) and properties (variables that belong to a class).

Instance

The easiset way to think of an instance, is to think of a copy. A class itself doesnt run, doesnt call methods, doesnt do anything. Similarly to the way that a cookie cutter itself cannot be eaten, it just stamps out instances of cookies in the shape they are supposed to represent

Polymorhism

The idea that different objects that adhere to the same interface can be interacted with the same way, but have different ways the accomplish the same behaviour or entirely diferent outcomes.

To put this in plain english, you have many different types of Vehicles. Trucks, Cars, 18 Wheelers, even a fourwheeler. They all share a subset of basic functionality though. They all can be start, stop, brake, drive, ect. The reason that we can think about these all the same, is we say that they share a common *interface.*

Interface

An interface is nothing more than a contract of methods / properties that a class must implement. That just says, if you're a class that uses X interface, then you have to have method, a, b, c and property, x,y,z. Interfaces are represented on the UML diagram with <<interfaceName>> and in code they often have names that begin with i, like IAnimal, ICar ect.

Inheritance ——▷

Classes can also inherit from other classes. When this happens they share the same properties and methods of their parents / ancestors. This is also known as subclassing. Inheritance is denoted in the UML diagram with an open arrow to the parent class. In the example to the left we say that the Lion is a subclass of Animal.

Aggregation

Aggregation is when one class is coposed of instances of other classes and those instances can be mutually exclusive. You can think of our example of a car for instance. A car has an engine, which itself can be an object, it has an on-board computer, tires, ect..

Composition

is when a class is made up other instances, but the class dependencies cannot exist without the other.

Delegation

The practice of proxying behaviour from one method to a different entity responsible for actually completing the action. Think of placing an update to your portfolio. Rather than making a trade yourself, you can call your broker who then deals with any underlying complexities and ends up carrying out the ultimate action for you.