Modelling Inheritance

- Inheritance is very powerful, but one of the most overused concept.

Is-a Relationship

- Inheritance is ‘Is-a’ or ‘kind-of’ relationship.
- “Here’s an abstraction and we want to create an abstraction that extends from that abstraction.”
- **Real reason** → to reuse the code that’s using an abstraction.

Base class or Super class

- There’s a base class or a super class, and there are derived classes, which are subclasses.
- The base class provides general behaviour, concepts, abstractions, and implementation, potentially.
- The derived class is the real specialisation.
- The external behaviour of the code should be consistent with the behaviour of the base class.

Inheritance is transitive

- If a class inherits from a base class and that class inherits from another base class, then the child class inherits from the ‘grandfather’ class.
- The derived object can be substituted for *any* class in the hierarchy.

Aggregation may be fixed, variable, or recursive

- Fixed aggregation → a wallet is a fixed aggregation. You own a wallet and that's fixed.
- Variable aggregation → the vehicles you own. You could buy another vehicle or get rid of one, etc.
    - Eg. a folder containing files.
- Recursive aggregation → you aggregate an object, which in turns aggregates another object, which in turn aggregates another object, and so on.
    - Eg. a folder contain a subfolder.
- Inheritance is static.

Generalisation vs. Inheritance

- Inheritance is usually at a class level.
- Generalisation is a generalisation of inheritance. Could be at a component level, class level, interface level.

Breadth vs. Depth

- Neither is good.
- **Inheritance should be as thin and short as possible**, rather than too broad or too deep.
- **Inheritance increases coupling**. If hierarchy is too deep, it becomes too hard to maintain this level of coupling. Something that changes toward the top could affect something much lower.
- If too broad, change in one class can affect many different classes.

Discriminator

- Difference between the base and the derived, and what makes one derived different from another is called discriminator.

- Eg. `Human` is the base class, the derived classes are `Man` and `Woman`. Gender becomes the discriminator.

Association vs. Aggregation vs. Inheritance

- Association → for objects of same stature. Object level relationship.
- Aggregation → ‘has-a’ or ‘part-whole’ relationship. Object level relationship.
- Inheritance → ‘is-a’ or ‘kind-of’ relationship. Class level relationship.

```
class Vehicle {}

class Engine {}

class Car extends Vehicle { // inheritance
        Engine engine; // aggregation
}
```

Abstract class

- A base class in which you cannot have objects/direct instances of this class.
- Purely used for extending and substitution purposes.
- Can have implementation.

Interface

- Higher elevation of abstract classes is an interface.
- **While abstract classes can have implementation in them, interfaces do not.**
- An interface gives much more extensibility by providing a contract with no implementation bindings.
- If you have an option between using interface and an abstract base class, use an interface.

Extension vs. Restriction

- When it comes to inheritance, as much as possible, we should extend rather than restrict.
- You shouldn't overwrite methods of a base class and do something completely different in behaviour than what's expected from the base class.
- We should be broadening not narrowing when we inherit.

Prototypal Inheritance

- It's where we use inheritance that isn't class based, but rather prototype based.
- A prototype is just another object. So, we inherit the features of another object rather than a class.
- JavaScript provides prototypal inheritance.