1. Composition vs Inheritance

Inheritance:

* What it is.
* When we define classes, we say it contains these properties and methods.
* Structure and structure of our code is around what it is, that is what these classes in our code consist of, we structure our classes around what things they have. Like, data, and methods that act upon that data.

Composition:

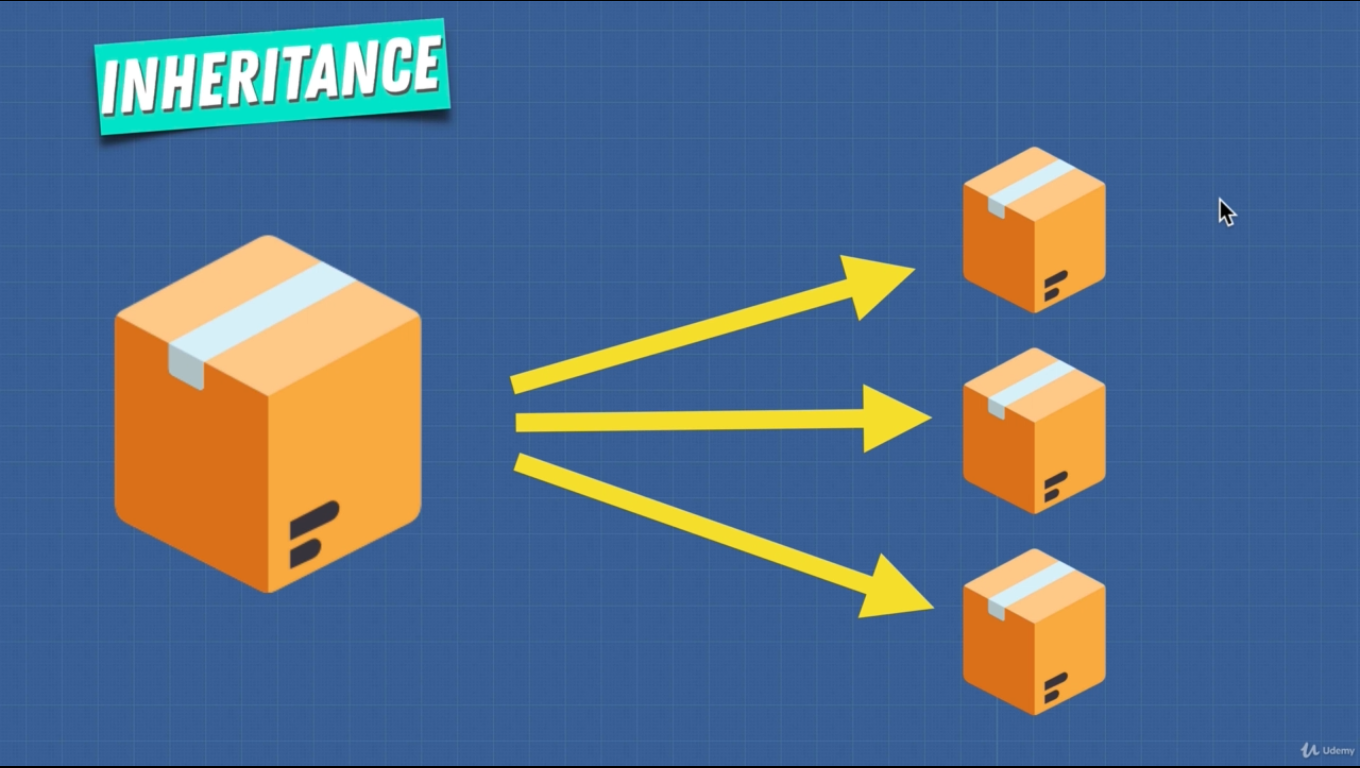
* What abilities it has.
* What it does to data.

Inheritance problem:

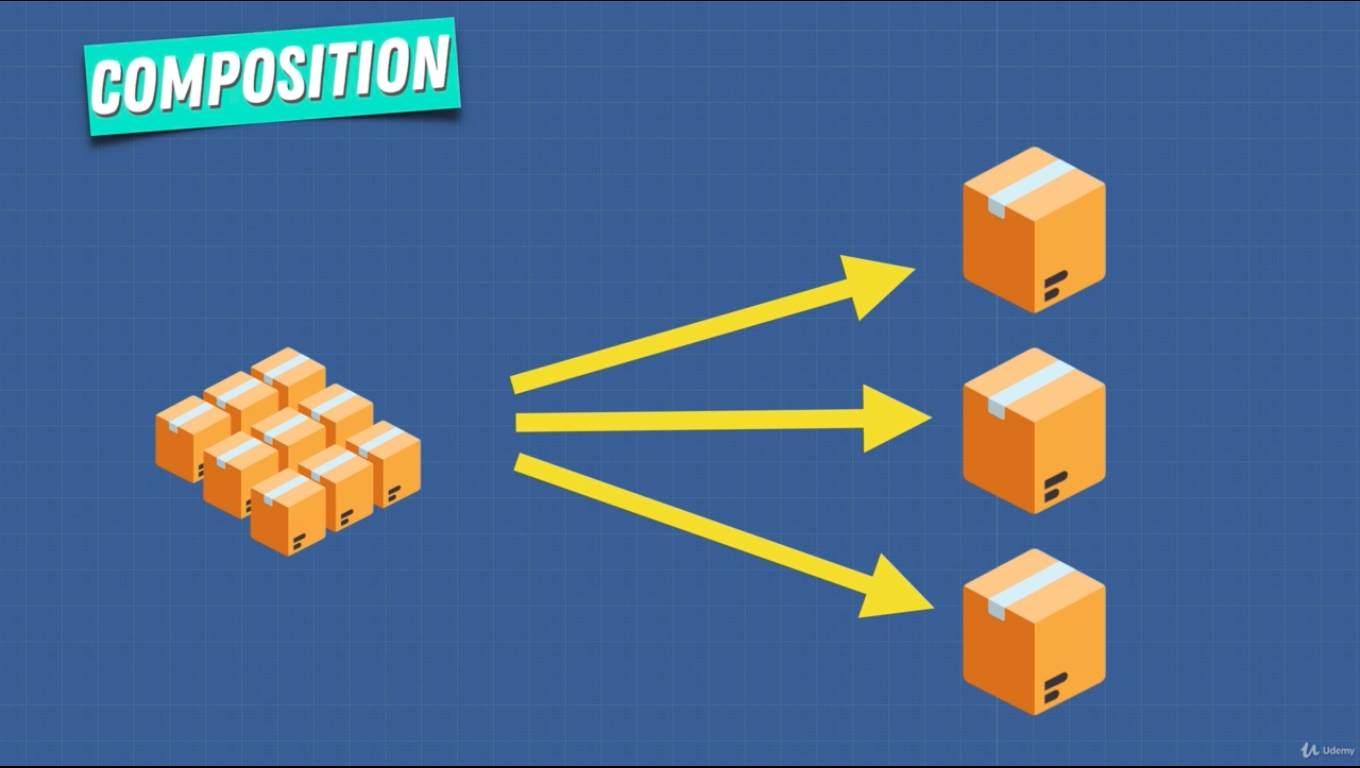
1. Tight Coupling problem: It is opposite of reusable modular code.
2. Tightly coupled inheritance can be good and bad at the same time.
   1. Good: We keep our code dry by having rippling effect that is we change in one place and the effect goes to other places.
   2. Bad: It has dependencies. This can break our code. May be other places don’t need that change.
3. Fragile base class problem: Changing the base class.
4. Hierarchy problem:
   1. We can have classes that inherits from other classes which is unnecessary for that lower class.
   2. Gorilla and banana problem. I need a banana you give me a gorilla holding a banana and the entire jangle underneath it.

We can fix these issues with composition.

* Removing all the methods.
* How to inherit from other class?
* The code is structed as what it does to the data instead what the class is.



Inheritance is inheriting properties from base class. Making smaller classes form bigger base class.



Composition is from small pieces we create bigger things.

* Composition is better for long term solution.

2. OOP vs FP

* Programming paradigm is writing code compliant with a specific set of rules.
* Organizing code to unit is OOP.
  + An object is a box containing info, and operations that are supposed to refer to the same concept or grouping it as an object.
  + These pieces of info inside of the objects are called attributes or state and the operation that can happen on the state are known as methods.
* Avoiding side effect and writing pure fn called FP.
  + Code is combination of functions.
  + Data is immutable.
  + No side effects.
  + Pure fn. Can not change the outside world.
  + Output value of fn simply depends on the given arguments.
* FP is based on different concepts.
  + HOF. Fn are first class citizen.
  + Pure fn.
  + Referential transparency.
* OOP
  + Objects are first class citizen.
  + Abstraction.
  + Encapsulation.
  + Inheritance.
  + Polymorphism.

FP:

* It is all about the idea of pure fn and composing fn to act upon the data.
* In FP fn manipulates DS like, trees, arrays, objects.
* Composition is used more than loops, conditions (these are more procedural).

The advantage of each paradigm is simply in the modelling of our algorithm and DS.

3. OOP vs FP 2

|  |  |
| --- | --- |
| FP | OOP |
| Many operations on fixed data. | Few operations on common data. |
| Stateless (no modification). | Stateful (modification) |
| Pure (no side effect outside of fn). | Side effect. |
| Declarative | Imperative |
| Good at processing large data. |  |
| Few things require many operations. | Many things few operations. |

* In all programming language we have data and behavior.

OOP says:

* Brings together the data and behavior in a single location called object or class.
  + Easy to understand
  + More organized.

FP says data and behavior are distinctly different things. They should be kept separate for clarity.