OOP Design

A Few Disclaimers

- **OOD questions are usually language agnostic**
- **However, they are based around languages like Java and C++**
- **JavaScript is OOP but in a different way**
  - Prototypal Inheritance and Prototype Chain
  - Has class syntax but is just syntactic sugar on top of the Prototype Chain
  - Privacy "does not exist traditionally" in JavaScript
    - In languages like Java, we get the private keyword
- **This doesn't mean you can't answer OOD questions**

OOP Design

- **A way to build programs**
  - Focuses on mapping "real world objects" to programs
  - Objects are the building blocks of OOP languages
- **A method for OOD thinking and analysis**
  - How does our system work/What are the goals of the system?
    - User stories
    - Use cases
  - What are the objects of the system?
  - What are the relationships between/among objects?

But Why?

- **Many companies still follow OOP principles**
  - OOP is still a part of the design of their systems
  - You'd be hard pressed to find a company that only does FP for example
- **It's a different perspective on how to build programs**
- **Think of classes/objects like React components before they were cool**
- **The idea is to be able to architect a system by thinking about all the moving parts and separating them into appropriate, logical components**
  - With system design, you're usually focused on the stuff that has to do with scale, schema design and API design; OOP tries to bring you closer client side

The Core Principles of OOP

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation

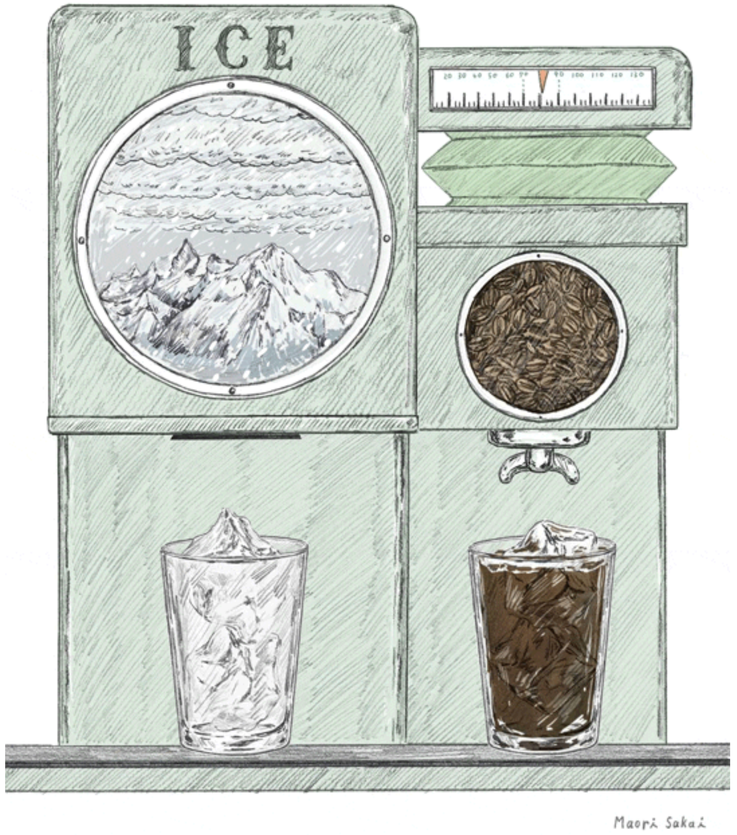
- **Encapsulation is the idea that we bundle data and methods together that work on that data**
  - Information hiding: We want to keep data/properties invisible to outside objects to prevent data tampering. This way, we can create methods that give only read access to our data.
    - Ex: In a person object, we might have their phone number as an attribute. This is probably not something a person might want to expose publicly. So, we can create a method called "getPhoneNumber", which can be called in certain instances (such as in the process to apply to Fullstack) to give "read access".

Abstraction

- **Abstraction is a logical result of encapsulation. Encapsulation can be seen as a method for abstraction. It means to hide everything but the relevant pieces/data about an object.**
  - If we reveal all information
    - Unsafe
    - Increases complexity
  - Ex: I like coffee. I have a coffee maker. I know how to use the coffee maker to make coffee. I don't need to know what the internal parts of the coffee maker do to make the coffee. The people who build coffee makers should care about those details.

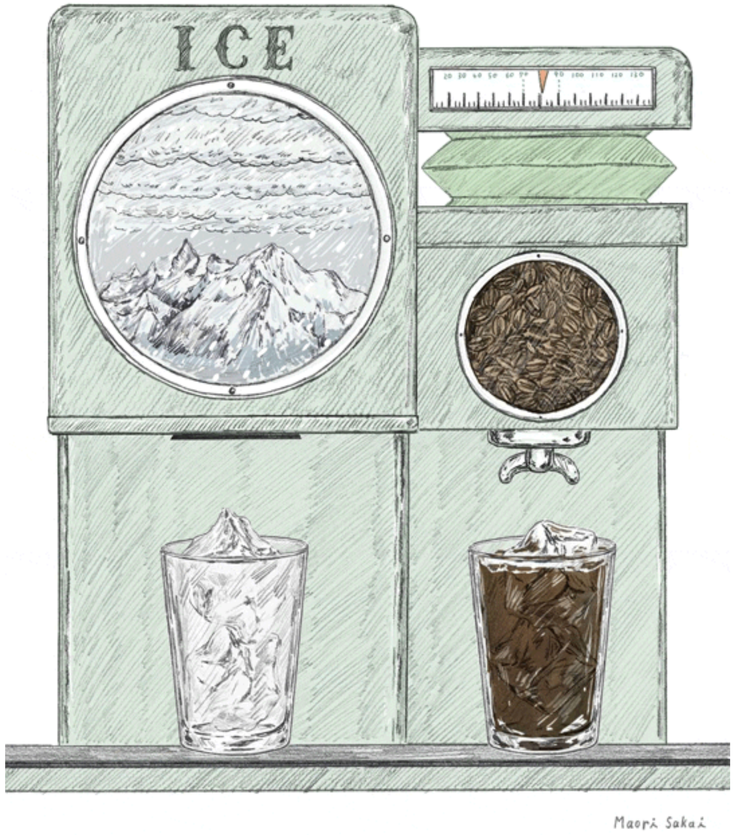
Inheritance

- **Inheritance is the idea that we can derive classes from other classes to create a hierarchy of classes that share a set of properties and methods**
  - Ex: All coffee makers should be able to make coffee (I hope). But we can have multiple types of coffee makers all with potentially different features but the core functionality remains the same.



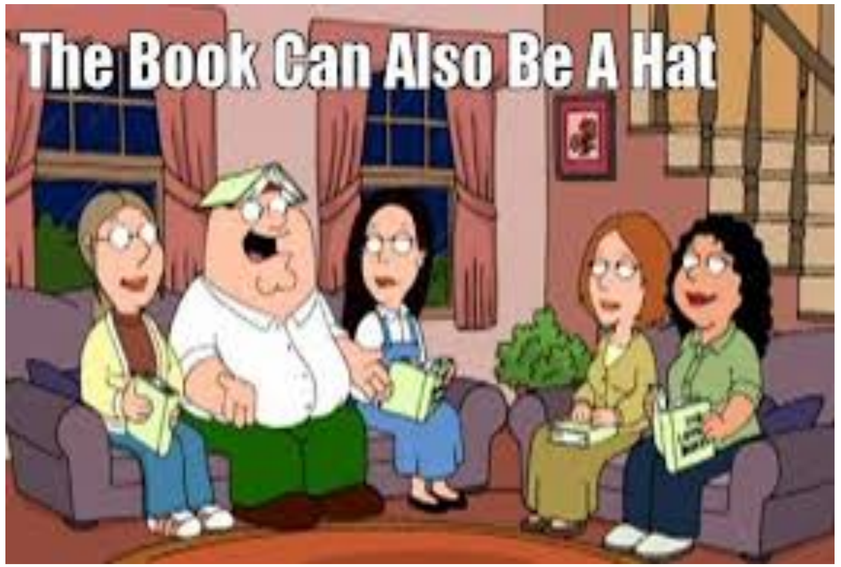
(Object) Composition Sidebar

- **Composition is the idea to combine (compose) objects to make larger, more complex objects**
  - This idea comes from functional programming (function composition)
  - Ex: All coffee makers have parts of it such as a spout, a coffee filter, (in this case an ice maker), some buttons that would not be as useful on their own as they would if they were put together



Polymorphism

- **Polymorphism is the ability of an object to take on many forms**
  - Ex: A general chess piece can take on many forms: Queen, King, Bishop, Knight, Pawn, Rook

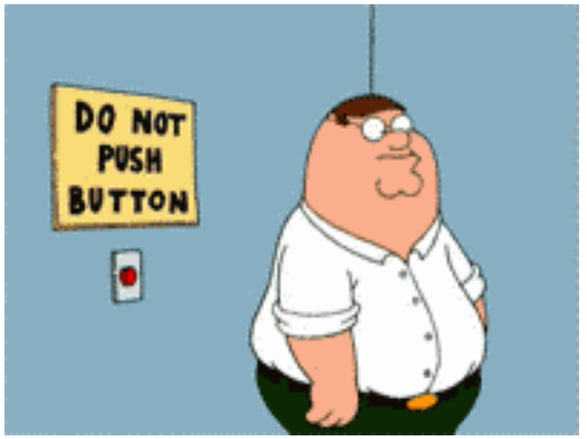


Planning Out an OO System

- **UML - Unified Modeling Language**
  - Similar to entity relationship diagrams
  - Has many subset diagrams
    - Activity Diagrams (User Stories)
    - Class Relationship Diagrams
  - Used for visualizing, constructing, and documenting systems

Activity Diagrams

- **Fancy phrase for user stories**
- **Takes the form a flow chart**
  - There is a certain set of actions that can be done with any system
  - We can define those actions as how we feel makes sense



Activity Diagram for a Basic Phone

Activity Diagram for a Coffee Maker

Class Diagrams

- **The many different components that comprise our system**
- **The relationship among those components**
  - As said before, think about it similarly to React Components. You wouldn't shove all your functionality into one component.

Class Diagram for a Basic Phone

Class Diagram for a Coffee Maker

Design a Deck of Cards and Black Jack

(Basic) Rules of Black Jack

- Players and Dealer get dealt separate hands with only two cards
- The goal is to make a hand as close to 21 points as possible without going over. If they go over, they fold
- The players and dealer have options to “hit” themselves to deal themselves another card to get closer to 21
- 2-10 have 2-10 values
- Jack, Queen, King all count as 10
- Ace can be counted as 1 or 11
- Two card winner is an Ace or a Jack, Queen or King