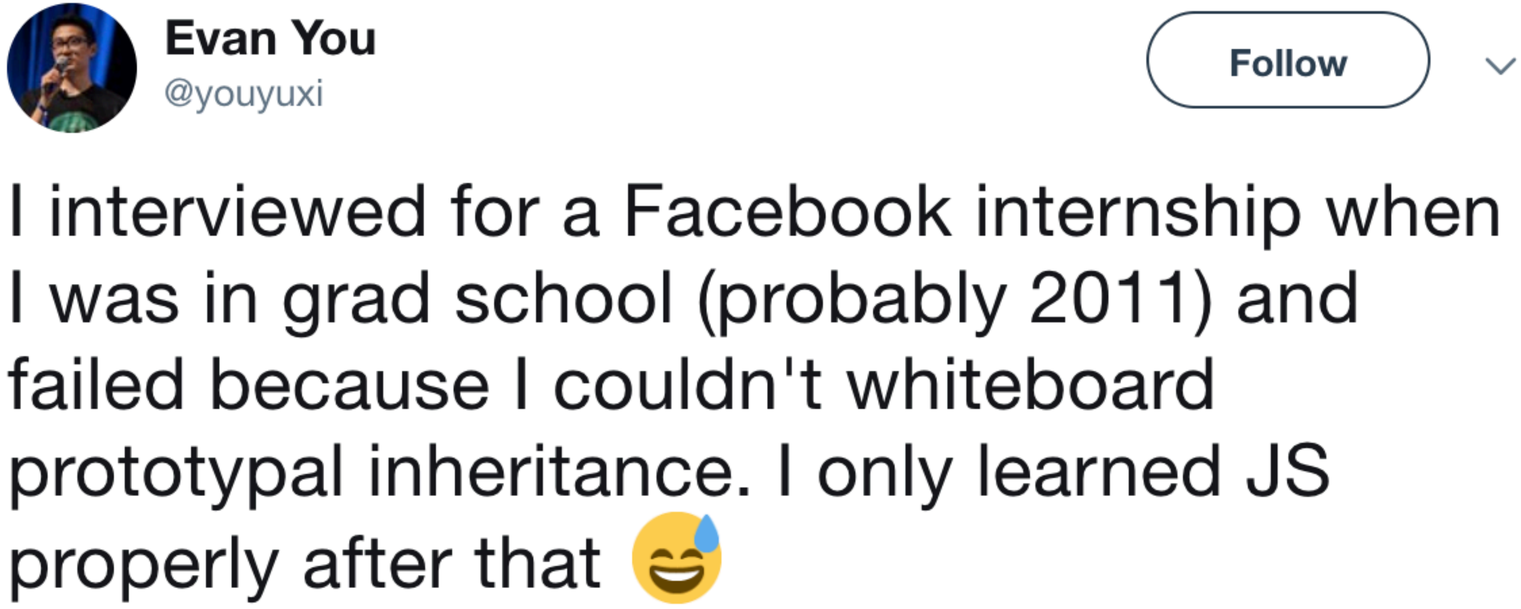
Prototypes vs. Classes
Model, View, ViewController

Class 4

Why learn prototypal inheritance?



Evan You is the author of one of today's big 3 JavaScript frameworks: Vue.js

Prototypal Inheritance

- Last night you all read: https://javascript.info/prototypes
- Prototypal Inheritance was JavaScript's predecessor to Class based inheritance (ES2015)
  - Concept first introduced in the Self programming language in 1987
- In 2019, your projects should choose Classes over Constructor Functions and Prototypes
- However, since Classes are implemented in terms of Prototypes and many existing libraries still make use of Prototypes, as a computer scientist front-end developer you should understand it.
- The gist is each object has a reference to a null-terminated linked list of other objects via their special [[Prototype]] property. If you access an object's property, and that property does does not exist, it searches up its prototype linked list for the first match.
  - Many more details, as covered in the text, but that's the big idea.
- You have more flexibility as a programmer in a prototypal inheritance than in traditional class inheritance
  - But history has proven this flexibility does not scale well (interoperability problems) and often leads to nuanced bugs

Questions on Prototypes?

Functions and `this`

- ***Methods* are *just functions*** defined on an object's prototype chain **in JS**.
- When you call a method using a *method call expression*, the special parameter *this* is established automagically by the language interpreter.
  - For example, **adaDog.speak()** is a *method call expression*.
  - In the example, the object is **adaDog** and the function is its **speak** property. When this method call is evaluated, **this** is **adaDog**.
- However, if you establish a *reference to the function* and invoke it using a *function call expression*, then this is unbound.
  - For example, **let speakFn = adaDog.speak;** establishes speakFn as a reference to adaDog.speak.
  - Then, calling **speakFn()** using a function call expression, assigns nothing to **this** in function body.
- Reference: https://javascript.info/object-methods

Refence: Closure

- As discussed on the whiteboard, JavaScript has native support for lexical closures. The following reference document explains closures in detail: https://javascript.info/closure

`call` and `apply`

- You can also *dynamically* bind this using a function's `call` and `apply` methods
- The `call` Method's first parameter is the this binding, subsequent are arguments
- The `apply` Method's first parameter is this binding, followed by an array of args
- Reference: https://javascript.info/call-apply-decorators

Example

- Let’s attempt to have a simple object method called one second after our page loads using setTimeout…

- We’ll be working in the 03-this example.

- Ways around *this* binding problems:

    1. Wrap the method call in an anonymous function (closure).

    2. Use the *bind* method of *Function’s* prototype.
       Reference: https://javascript.info/bind

Model - View - ViewController

- Let’s work with example 04 in code to put together an image gallery!