Web Programming

Week 7

"Classes tend to be bad modules."

D. Crockford, How JS works, p. 17.0

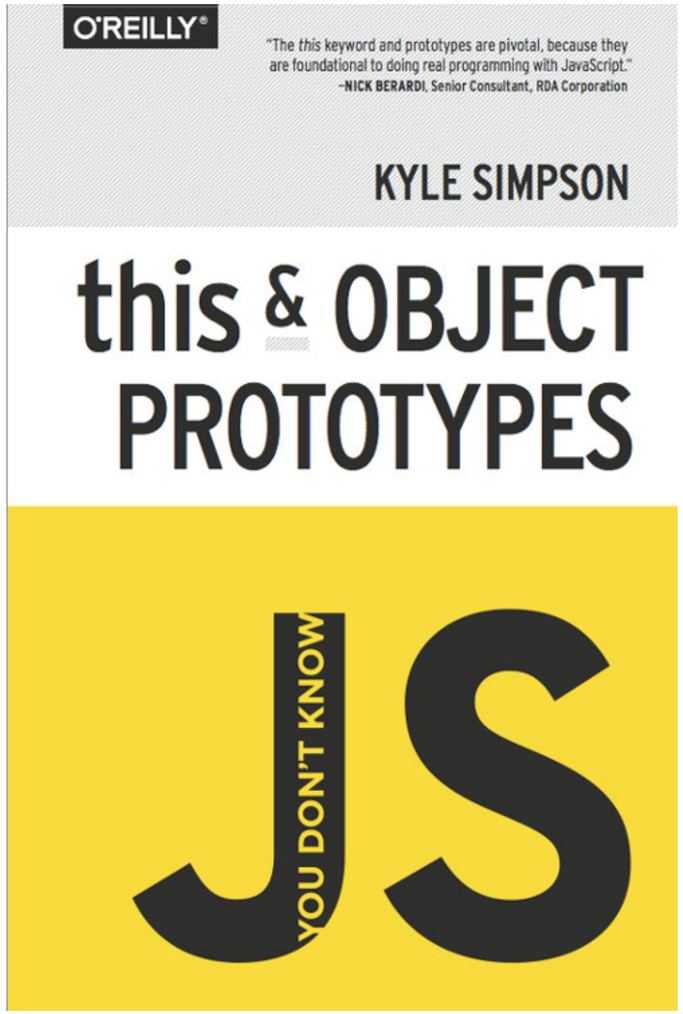
Today: Classes

Keywords: class, extends

"Inheritance" through Prototypes

Dynamic Dispatch, Refactoring Tests

<Exkurs/>, Quiz



Refresher

https://github.com/getify/You-Dont-Know-JS

Three ways to encode objects.

Open, dynamic

JS "Objects"

```
const good = {
    firstname : "Good",
    lastname  : "Boy",
    getName   : function() {
        return this.firstname + " "  + this.lastname
    }
};
// no safety but super dynamic
// unobvious how to share structure
// beware of "this"! See Adam Breindl last week.
```

Closed, explicit

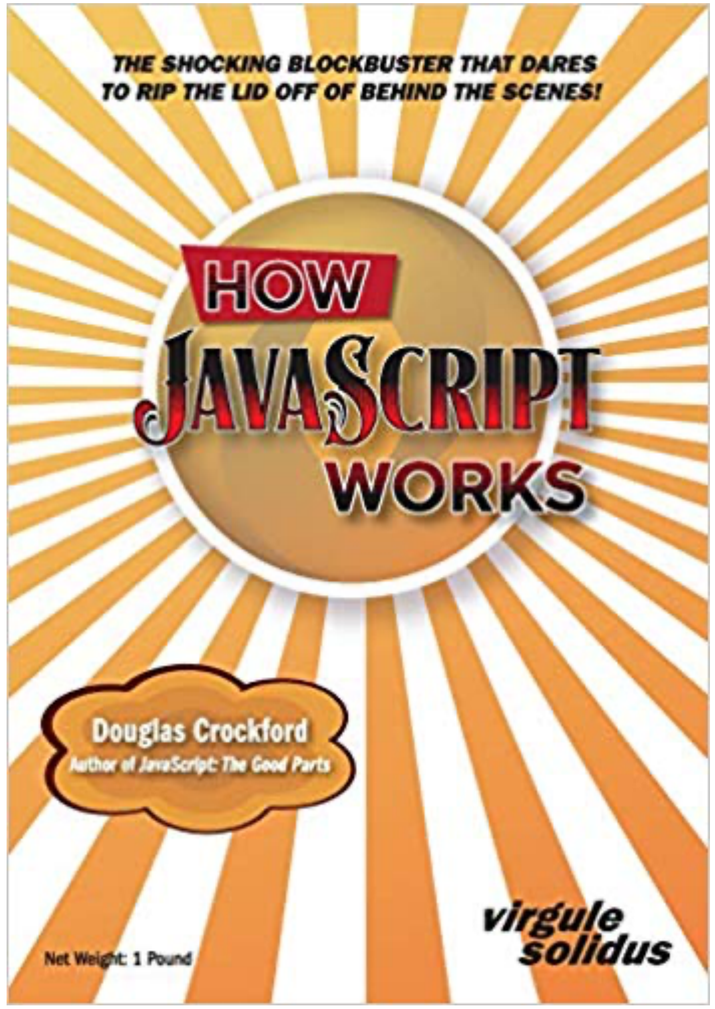
closure scope, no "this"

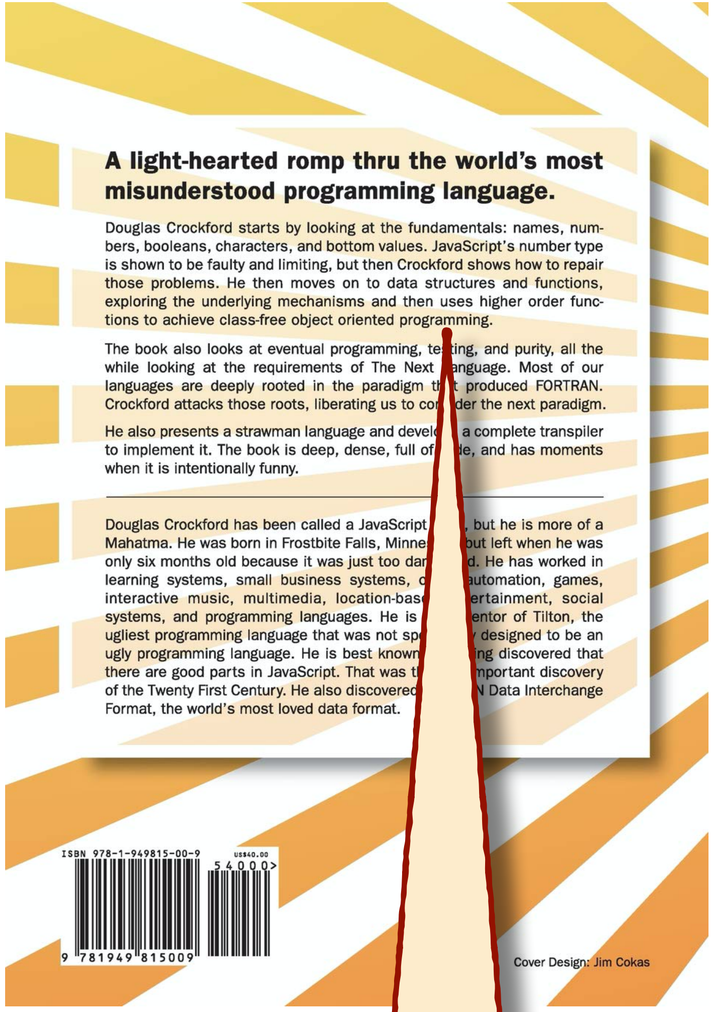
```
function Person(first, last) {
    let firstname = first;  // optional
    let lastname  = last;
    return {
        getName: function() {
                  return firstname + " "  + lastname }
                 }
    }
}
// best safety, easy to share structure, but no class
```

Mixed, classified

depends on "new"

```
const Person = ( () => { // lexical scope
    function Person(first, last) { // ctor, binding
        this.firstname = first;
        this.lastname  = last;
    }
    Person.prototype.getName = function() {
            return this.firstname + " " + this.lastname;
    };
    return Person;
}) (); // IIFE
// new Person("Good", "Boy") instanceof Person
```





those problems. He then moves on to data structures and functions, exploring the underlying mechanisms and then uses higher order functions to achieve class-free object oriented programming.

class Keyword

Syntactic sugar for variant 3
(mixed, classified)

Since ES6

class Example

close to version 3

```
class Person {
  constructor(first, last) {
    this.firstname = first;
    this.lastname  = last
  }
  getName() {
    return this.firstname + " " + this.lastname
  }
}
// new Person("Good", "Boy") instanceof Person
```

extends Keyword

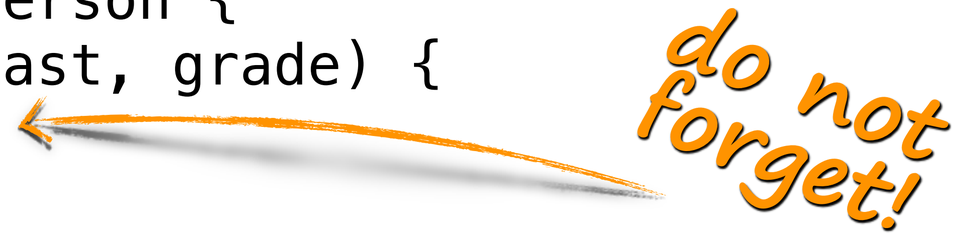
Syntactic sugar for creating a prototype chain.

Since ES6

extends Example

```
class Student extends Person {
  constructor (first, last, grade) {
    super(first, last);
    this.grade = grade;
  }
}

const s = new Student("Top","Student", 5.5);
```



Functions are Objects

They have the prototype property.

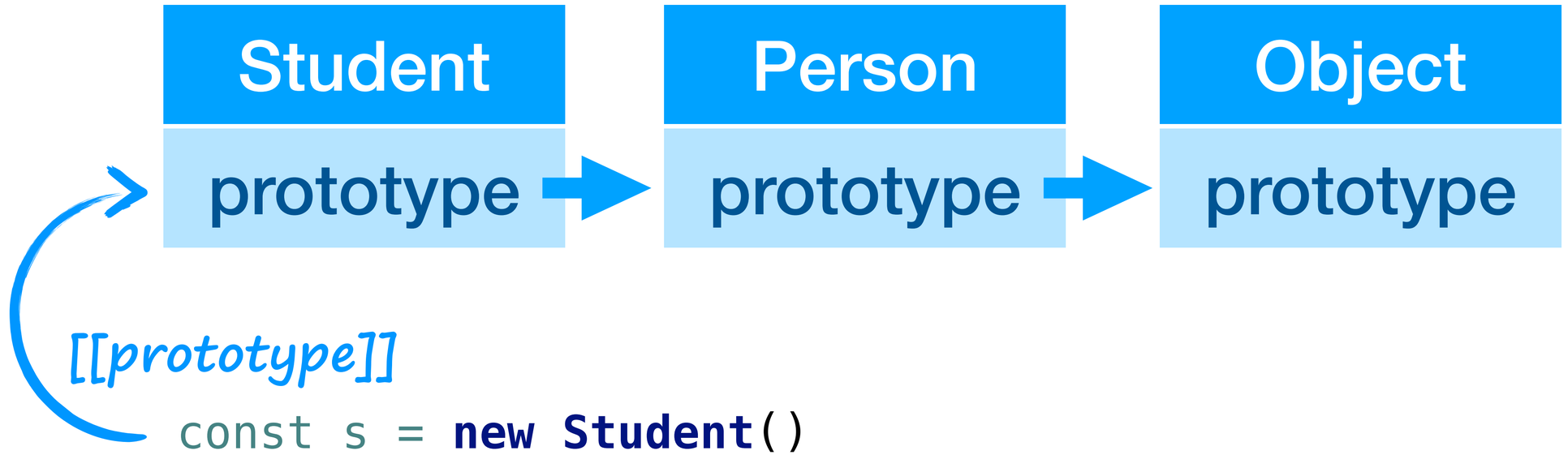
It references an object that
- has a name (~ "Type")
- has a constructor (Type Function)
- has itself a prototype

Objects are Functions

Not in the sense of JavaScript functions.

But in the sense of computer science, they are functions (in the general sense) from their keys to their values.

Prototype chain




```
const s = new Student()
// s.__proto__                  === Student.prototype;
// Object.getPrototypeOf(s) === Student.prototype;
// => s instanceof Student
```

Prototype ++

Since a prototype is an object,
it can be modified and extended

One can replace the prototype at
runtime, essentially changing the "type"

Object.setPrototypeOf(obj, proto);

Dynamic Dispatch

Properties (and so also functions) are first looked for in the object and then in its prototype.
And since prototypes are themselves objects, their prototype are used as well - making a chain until Object.prototype.

This looks like inheritance.

Let's test!

https://babeljs.io/repl/

Idea

```
(10).times( n => console.log(n) );
const squares=(10).times(n=> n*n);
```

Can we do this?

Dispatch

Java — Static, based on the static type

Groovy — Dynamic, based on the runtime type

JavaScript — Dynamic by name,
Pattern: Chain of Responsibility

Thought Experiment

What's the purpose of classes and inheritance in a language that does not use types in the method dispatch?

Examples? Counter-examples?

Classes - Abstraction

Abstractions can be classes - or just functions.

For the abstraction, does it make a difference, whether **Student** is a **Class** or a **Function**?

Inheritance I - Polymorphism

This concept does not apply to JS since method dispatch is independent of the argument type.

Inheritance II - Sharing

Superclasses can provide implementations for subclasses.

Are there alternatives?
(see homework)

Prepare at Home

Install node.js (recent version)

Install npm (recent version)

Work at Home

Reading: Eric Elliott

https://medium.com/javascript-scene/composing-software-an-introduction-27b72500d6ea