Communism OOP

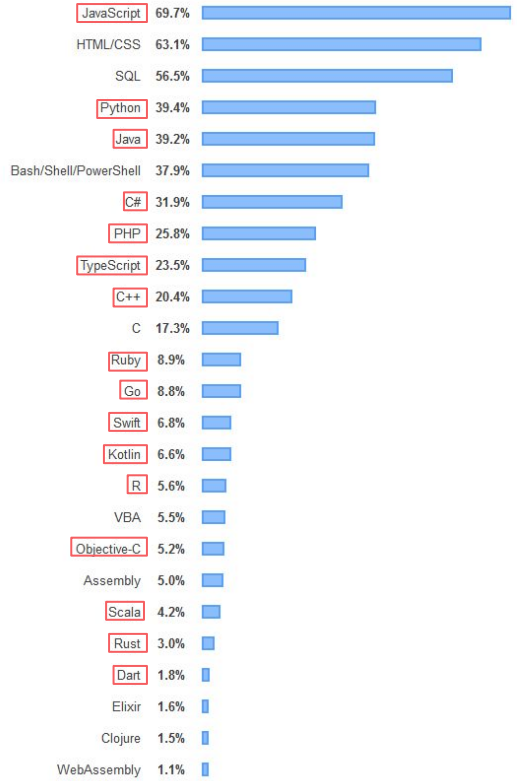
**For world without classes,
Objects of the world,
UNITE!**

What is Object-oriented Programming?

What is Object Oriented Programming?

- Object Oriented Programming (a.k.a. 객체지향 프로그래밍)

- 프로그래밍 언어 예시?
  - C++
  - C#
  - Java
  - etc.



How do you define Object-oriented Programming?

What many instructors teach us

객체지향 == 클래스

클래스는 **청사진**(Blueprint)과 같은 것

클래스를 이용해 객체를 만들면 객체지향이다!

이런 거

……과연 그럴까?

~~(물론 시험 때는 배운 대로 안 쓰면 점수 망해요)~~

The Fantastic Four (and no, not *the* F4)

만약 어떤 언어가 아래 조건을 만족한다면, 객체지향 언어라고 할 수 있다.

1. **Encapsulation** (캡슐화)
2. **Abstraction** (추상화)
3. **Inheritance** (상속)
4. **Polymorphism** (다형성)

Q. 에게, 이게 다예요?

A. 네, 저게 다예요.

(킹갓겜이니어서 구매하고 광명을 찾으십시오)

Untitled Goose Game by *House House*, 2019

Goose honks at you, steals your sh*t, and...

거위의 현재 상태

- 무전기를 훔쳐서 입에 물고 있음
- 꽥 소리를 지르고 있음
- 인간에게 크고 아름다운 엿을 먹이고 있음

거위 객체

- 입에 물고 있는 물건, 고개를 숙였는지 여부
- 꽥 소리지르기, 물건 잡아당기기, 도망가기

⇒ Encapsulation

Goose honks, steals, bites, ducks, runs...

실제 거위의 특징

- 색, 울음소리 톤, 성별, 나이, 이름, 먹은 것…
- 물기, 알 품기, 날기, 걷기, 달리기, 먹기…

게임에서 필요한 거위의 특징

- 입에 물고 있는 물건, 고개를 숙였는지 여부
- 걷기, 달리기, 울기, 물기, 당기기

⇒ Abstraction

Oh, those poor, poor humans...

인간의 공통점
- 앉기, 서기, 달리기, 물건 집기

아이(인간)의 특징
- 거위가 근처에서 울면 도망감 쫄보쉑 ㅉㅉ

아이의 부모(인간)의 특징
- 거위를 무서워하지 않고 잡으러 옴

⇒ Inheritance

I am inevitable (Goose is, not me)

이 스크린샷에서 볼 수 있는 것들
- 하모니카, 링, 모자…

조금만 더 뒤로 나가면,
- 샌드위치, 피크닉 상자, 리본, 거위 동상…

공통점이 뭘까?
□ 거위가 입에 물 수 있는 물건

⇒ Polymorphism

Thanks, Goose, now what?

클래스 기반 객체지향 언어는 이 조건을 만족하는가?

1. 캡슐화 — 클래스 선언, public / private 멤버 선언
2. 추상화 — 클래스 선언
3. 상속 — 상속 (...), 서브클래스 선언
4. 다형성 — 인터페이스 기반 서브타이핑

“아니, 근데 클래스 없이 이걸 어떻게 한단 말이야?”

Classless Object-oriented Programming

All objects are equal, and must remain equal

사실 객체 지향에 클래스가 가지는 의미는 **None** 어떻게 보면 오히려 객체지향의 개념을 조금 이해하기 어렵게 하기도

객체지향의 필수요소로부터 도출되는 객체지향의 *지향점*

1. 모든 것은 객체이다.
2. 모든 오브젝트는 서로 **메시지**를 주고받아 통신한다.

클래스는 그저 또 다른 타입에 지나지 않는다

Yes, everything is an object; so are you

"자, 재밌는 상상 한 번 해 보자고."

- 하정우, 모 ~~똥망~~게임 광고에서

- 만약 클래스도 객체라면?
  - 클래스도 인스턴스와 같은 객체이다
  - 그럼 클래스는 어떤 클래스의 인스턴스…?
    - Metaclass (Python, Smalltalk-80)
    - Eigenclass (Ruby)
- 애초에 클래스 자체가 없다면?
  - 객체는 객체 그 자체로 존재
  - 청사진(Blueprint)? 그게 뭐야, 먹는 거야?
  - "아니, 그럼 상속과 다형성은 어떻게 구현해?"

딱 기다리세요, 보여드릴게

재밌는 상상 한 번 해 보자고 꼬시는 하정우씨
~~으악 망겜 안해요~~

Longing for a class-less society

- 여기서는 *클래스도 객체인 경우*는 다루지 않음
  - “근데 왜 메타클래스 차별해요? 이건 **메혐**이야!” (그치만 그렇다고 호날두클래스라고 부를 순 없잖아요?) (미안합니다)
  - Correction: 메타클래스를 차별하는 것이 아니라 클래스에 의한 차별을 차별하는 것 (?)
  - Reason: 이 발표의 주제는 클래스 없는 이상적인 사회를 그리는 것이기 때문 (????)

- 클래스 없는 객체지향 패러다임?

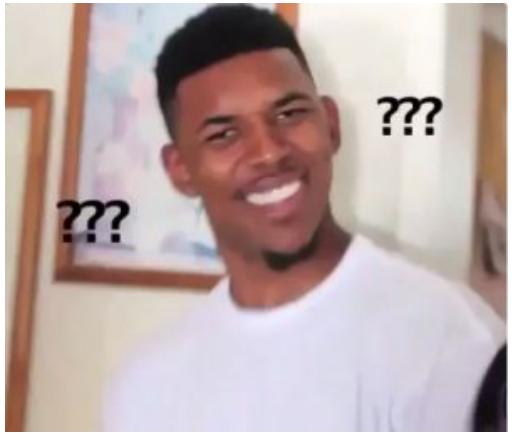
Prototype-based Object-oriented Programming

Here comes a new challenger!

프로토타입 기반 객체지향 프로그래밍

- 모든 객체는 **슬롯**을 가지고 있다
- 이 **슬롯**에는 데이터, 함수 등이 들어갈 수 있다
- 객체는 **빈 객체(null object)**에 슬롯을 추가함으로써 만들어진다
- **객체**는 **객체**일 뿐, 그 이상도, 이하도 아니다

- 상속은 **프로토타입**으로 구현한다
- 다형성은 보통 **덕 타이핑**으로 구현한다



With classes, everything is possible

클래스는 객체를 만들어내는 청사진(blueprint)

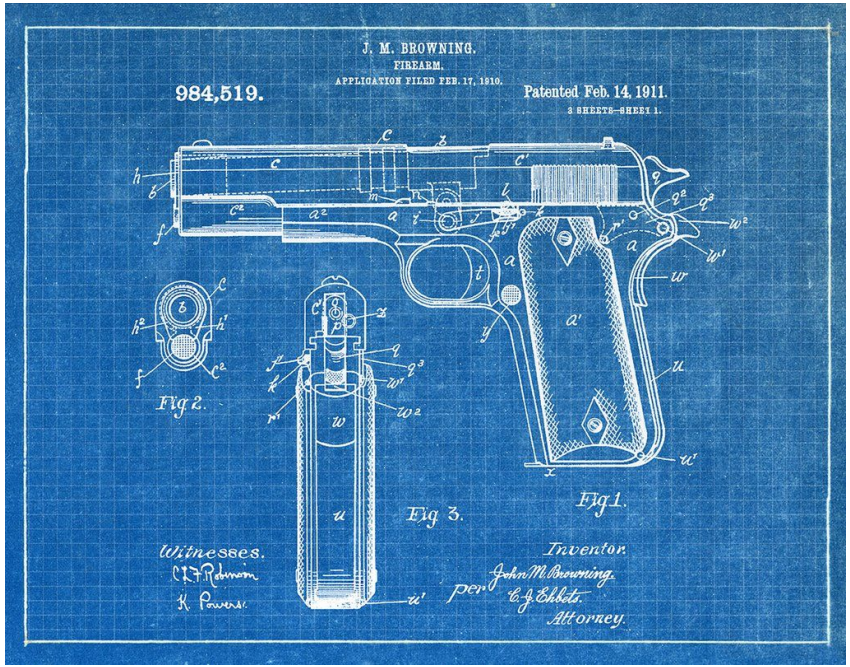
같은 청사진을 이용하면 같은 객체가 나옴

ex) M1911의 청사진으로 건물을 지을 수는 없음 (당연하지만)
물론 그 다음 어떻게 객체를 써먹을지는 내 마음

만약 원래 청사진에서 조금 기능을 바꾸고 싶을 때?
⇒ 원래 청사진을 **상속**해서 기능을 추가!

ex) M1911을 개량해서 M1911A1을 제작

여기서 청사진을 상속하는 것은, 기존에 존재하는 청사진을 바탕으로 **새 청사진을 만드는 것**



두루두루 사랑받는 명품 피스톨, Colt M1911

With prototypes, nothing is to fear

이것은 상자입니다. ~~사람 한 명 쯤 거뜬히 넣을 수 있죠.~~

"ex nihilo" (La. *무(無)로부터*)

객체는 데이터를 담는 상자

꼴랑 상자 하나를 만드는 데 청사진은 필요하지 않음

슬롯(slot)은 객체라는 상자에 들어가는 물건(data)

1. 상자를 만들어서 네임펜으로 이름을 적는다
2. 들어갈 물건에 포스트잇으로 이름을 적는다
3. 만들어둔 상자에 물건을 집어넣는다
4. ???
5. PROFIT!

만약 원래 상자의 내용물을 조금만 바꾸고 싶을 때?
⇒ 새 상자를 만들고 **원래 상자를 가리키게** 하자!
⇒ 원래 상자를 가리키는 포인터가 **프로토타입**

Is that a goose... I mean, duck?

클래스 기반 프로그래밍의 경우, 클래스 자체를 타입이라 해석할 수 있음 struct처럼요!

ex) `class Animal { };` `class Dog : Animal { };` `class Cat : Animal { };`

⇒ **타입**의 개념으로 다형성을 구현

그렇지만 프로토타입 기반 프로그래밍은…? 애초에 객체의 타입이라 부를 만한 게 없죠

덕 테스트(duck test)

"If it walks like a duck, and it quacks like a duck, then it must be a duck."

⇒ 덕 타이핑(duck typing)

OK, enough theories; show me the *real* deal

The Original

Self

처음으로 프로토타입 기반 객체지향을 제시한 언어
⇒ 프로토타입의 할아버지

Smalltalk-80을 기반으로 하고 있음

쓰이는 일은 거의 없음
근데 누가 이런 언어로 코딩하려고 써요 다 Proof of Concept지 뭐

```
"우선 동물 객체를 만들고"
_AddSlots: (|
    animal <- (| parent* = traits cloneable |)
|).
animal _AddSlots: (| name |).

"동물 객체로부터 개 객체와"
_AddSlots: (| dog <- (| animal copy |) |).
dog name: 'Doggy'.
dog _AddSlots: (| cry = (| | 'Woof!' print.) |).

"고양이 객체를 만듭니다."
_AddSlots: (| cat <- (| animal copy |) |).
cat name: 'Kitty'.
cat _AddSlots: (| cry = (| | 'Meow!' print.) |).

dog cry      "--> Woof! 출력"
cat cry      "--> Meow! 출력"
```

OK, enough theories; show me the *real* deal

The Outlier

Lua

테이블을 이용해서 객체지향을 구현하는 것이 특징

모든 것은 ~~객체~~ 테이블이다!

C 또는 C++에 붙여서 쓰는 "애드온" 언어

보통 게임 애드온 만들 때 많이 쓰임 네, GTA의 그거예요

```lua
-- 우선 동물 객체를 만들고
Animal = { name = "" }

-- 동물 객체로부터 개 객체와
Dog = { name = "Doggy" }
setmetatable(Dog, { __index = Animal })
function Dog.cry ()
	print("Woof!")
end

-- 고양이 객체를 만듭니다.
Cat = { name = "Kitty" }
setmetatable(Cat, { __index = Animal })
function Cat.cry ()
	print("Meow!")
end

Dog.cry() --> Woof! 출력
Cat.cry() --> Meow! 출력
```

OK, enough theories; show me the *real* deal

The Enforcer of Web

JavaScript

가장 널리 알려진 프로토타입 기반 언어

함수를 사용해서 객체를 만드는 **생성자 패턴**이 흔하다는 것이 특징

웹을 쓰는 사람이라면 한 번 쯤은 만져봤을 것

객체 리터럴을 사용해서 객체를 만들 수도 있음

ex) `var Coordinate = { x: 0, y: 0, dist: function() {...} };`

```
var Animal = function() { this.name = ""; };

var Dog = function() {
    this.name = "Doggy";
    this.cry = function() { console.log("Woof!"); };
};
Dog.prototype = new Animal();

var Cat = function() {
    this.name = "Kitty";
    this.cry = function() { console.log("Meow!"); };
};
Cat.prototype = new Animal();

var dog = new Dog();
dog.cry();    // Woof! 출력

var cat = new Cat();
cat.cry();    // Meow! 출력
```

At the end of the day...

프로토타입 기반의 **장점**

- **새 오브젝트**를 만드는 것이 쉬움
- 한 오브젝트가 **다른 오브젝트를 만드는 데 쓰일** 수 있음
- **다중 상속**의 구현이 비교적 자유로움
- **런타임에 프로그램의 동작을 바꾸는 것**이 가능함

프로토타입 기반의 **단점**

- **실행 속도**가 느림
- **리팩토링이 힘들어질** 가능성이 있음
- **디버깅도 힘들어질** 가능성이 있음

For those with unlimited possibilities

만약 객체지향을 오늘 **처음** 접했다면?

- 객체지향은 **모든 것을 객체**라고 생각하는 프로그래밍 방법론
- 클래스 기반과 프로토타입 기반은 두 가지 **다른 접근 방법**

Q: 뭐가 더 좋아요?

A: 나도 몰라요, 직접 해 보세요.

객체지향의 기본만 이해해줘도 저는 기뻐요♪

For those with *a priori* knowledge

만약 객체지향을 이전에 **이미** 접해봤다면?

- 클래스와 프로토타입 사이의 **구현 방식 차이**
- **머리를 조금 말랑말랑**하게 해서 보면 이해가 잘 될 거예요
- 공장에서 찍어내는 **공산품** vs. 기존에 존재하는 것을 복사하는 **수제 용품**
- Trial and Error!

언젠가는 여러분도 객체지향을 사랑할 수 있게 될 거예요♪

The very last TMI(Too Much Information)

프로토타입 기반 객체지향은 **모든 것은 객체**라는 절대적 진리에 가장 근접한 방법

동적으로 문제를 해결할 수 있다는 것이 매우 매력적

와! 다이나믹! 멋져!

Lua로부터 많은 영향을 받은 언어 디자인

~~그리고 JavaScript는 똥망언어에요~~

Thank you.