당신이 몰랐던 객체지향 Multiple Dispatch

이정연

BBConf Winter 2025

On Design Pattern

디자인 패턴 좋아하세요?

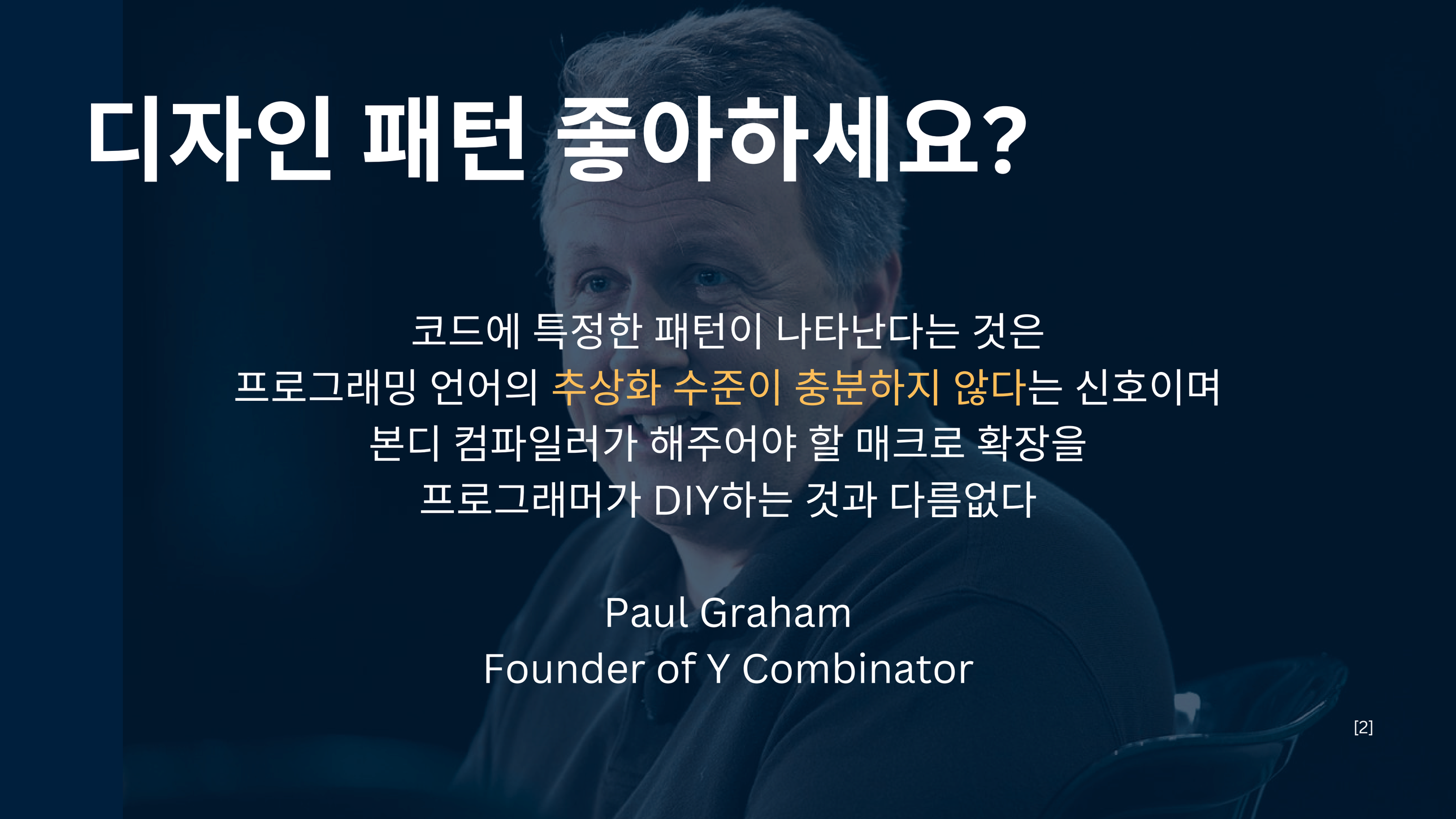
디자인 패턴 좋아하세요?

당신이 디자인 패턴이라고 부르는 것의 실체는
당신의 프로그래밍 언어에 대한 버그 리포트이다

Peter Norvig
(prev.) Research Director @Google

[1]

디자인 패턴 좋아하세요?



코드에 특정한 패턴이 나타난다는 것은
프로그래밍 언어의 추상화 수준이 충분하지 않다는 신호이며
본디 컴파일러가 해주어야 할 매크로 확장을
프로그래머가 DIY하는 것과 다름없다

Paul Graham
Founder of Y Combinator

[2]

디자인 패턴:
숭고히 지켜야 할 유산인가?

- 서브루틴: 스택 프레임을 구현해 호출 추적을 가능하게 하는 **패턴의 이름**
  - 함수의 복귀 주소가 정적으로 지정되었던 포트란 시절
- 객체지향 클래스: 구조체에 함수포인터를 수동으로 바인딩하는 **패턴의 이름**
  - 클래스와 객체 바인딩이 없는 C가 주류였던 시절
- 뮤텍스, 배리어, 랑데부: 세마포어를 이용한 동기화 **패턴의 이름**
  - 다익스트라가 세마포어를 발명한 직후
- GoF의 디자인패턴: C++, Java의 문제를 해결하는 20여가지 **패턴**들
  - 디자인 패턴의 왕국은 영원할 것인가?

[3][4]

On Single Dispatch

카카오톡 모가지:

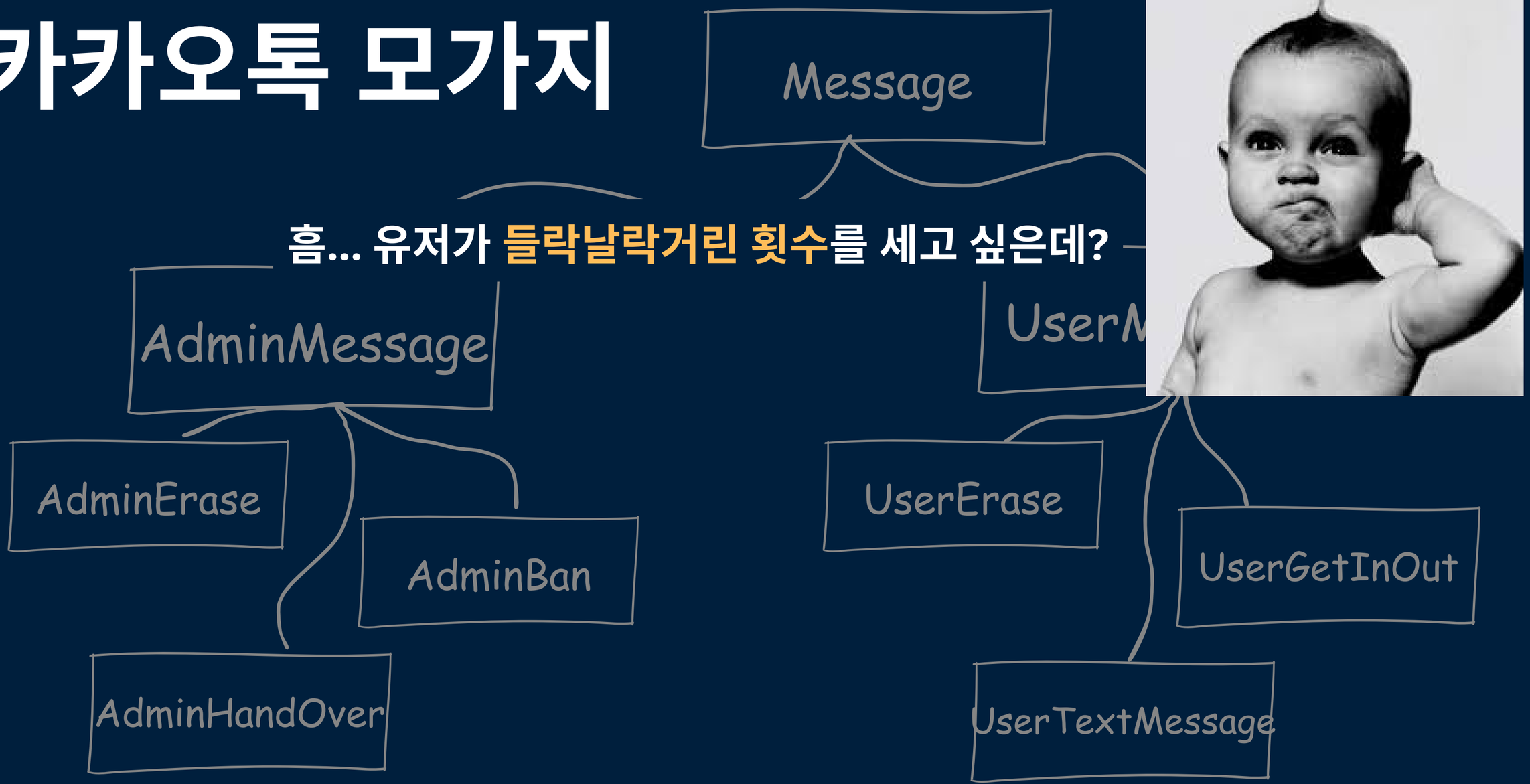
백준푸는방 오랜 전통의 유틸리티

- 채팅방 통계를 내서 유저를 정리하는데 씁니다.
  - 가장 많이 떠든 사람
  - 가장 적게 떠든 사람
  - 가장 많이 밴을 당한 사람
  - 활동한 유저들의 목록
  - 등등...
- 다양한 멤버들에 의해 다양하게 재구현 됨

카카오톡 모가지


Message
AdminMessage
UserMessage
AdminErase
AdminBan
AdminHandOver
UserErase
UserGetInOut
UserTextMessage

카카오톡 모가지


Message
흠... 유저가 들락날락거린 횟수를 세고 싶은데?
AdminMessage
UserM
AdminErase
AdminBan
AdminHandOver
UserErase
UserGetInOut
UserTextMessage

카카오톡 모가지


Message
AdminMessage
AdminErase
AdminBan
AdminHandOver
UserM
UserErase
UserGetInOut
UserTextMessage

아하!

이렇게만 된다면 행복하지 않을까요?

```
class Mogazi {
    int doCount(List<Message> messages) {
        var counter = new GetInOutCounter();
        for (var message : messages) {
            counter.count(message);
        }
        return counter.count;
    }
}
```

```
class GetInOutCounter {
    int count = 0;

    void count(AdminBan message)     { count += 1; }
    void count(UserGetInOut message) { count += 1; }
    // 일반적인 경우
    void count(Message message)      { return;     }
}
```

*물론 이 문제를 해결하는 여러가지 다른 많은 방법이 있습니다.
하지만 여러분이 처음 떠올린 방법이 이것인 경우를 가정해봅시다

오버로드 메서드 바인딩은 정적입니다

```
class Mogazi {
    int doCount(List<Message> messages) {
        var counter = new GetInOutCounter();
        for (var message : messages) {
            counter.count(message);
        }
        return counter.count;
    }
}
```

```
class GetInOutCounter {
    int count = 0;

    void count(AdminBan message)     { count += 1; }
    void count(UserGetInOut message) { count += 1; }
    // 일반적인 경우
    void count(Message message)      { return;     }
}
```

뭐야, 이거 왜 안 돼?

- 혹시 될 지도 모른다고 생각하셨나요?
  - 괜찮습니다! 여러분이 생각한 코드는 지극히 정상적이고 직관적입니다
  - 자바 개발자들이 무슨 의도가 있어서 일부러 막아둔 게 아닙니다
- ‘객체지향은 원래 그런거야! 그러게 방문자(Visitor) 패턴을 공부했어야지!’
  - 글쎄요... 과연?

On Multiple Dispatch

사실 다중 디스패치는 초기부터 논의되어왔습니다

- “이런경우엔 어떻게하죠?”
  - “음.. 그냥 디스패칭을 두번 하면 되지 않을까?”
- 비지터패턴: 디스패칭을 두 번 하는 **패턴의 이름**

The Double Dispatch Pattern

- Problem: behavior depends on two different classes

Result Type for Addition Operation

Right Operand

Left Operand	Integer	Fraction	Float	Complex
Integer	Integer	Fraction	Float	Complex
Fraction	Fraction	Fraction	Float	Complex
Float	Float	Float	Float	Complex
Complex	Complex	Complex	Complex	Complex

왜 디스패치를 한 번만 해야 하지?:
멀티플 디스패치

```
class GetInOutCounter {
    int count = 0;

    void count(AdminBan message)      {
    void count(UserGetInOut message) {
    // 일반적인 경우
    void count(Message message)       {
}
```

```
class Mogazi {
    int doCount(List<Message> messages) {
        var counter = new GetInOutCounter();
        for (var message : messages) {
            counter.count(message);
        }
        return counter.count;
    }
}
```

이게 되지 말아야 할 이유가 있나요?

그런데 이거 어디에 쓰나요?

- 멀티메서드는 확장가능한 런타임 패턴매칭입니다
  - 확장메서드: C#, Kotlin, etc...
  - 패턴매칭: Rust, Haskell, Scala etc...
- 여러분의 도구상자에 if문을 줄여주는 또 하나의 도구를 추가하세요
  - Visitor Pattern, Builder Pattern을 대체할 수 있습니다
- 멀티메서드가 뭐가 그렇게 특별하죠?
  - 멀티플 디스패치는 싱글 디스패치의 일반화입니다
  - 싱글 디스패치로는 가능하지 않았던 많은 것들!
- 근데 우리 언어에는 그런게 없는데...

멀티메서드를 구현한 언어들

- Groovy: built-in
- Common Lisp: built-in
- Clojure: built-in
- Julia: built-in
- Dylan: built-in
- C#: as dynamic 키워드를 통한 부분적 지원
- Python, Javascript, Rust, Kotlin, PHP, etc.: 라이브러리가 있음
  - 자바는 없습니다

자료의 출처

1. 모든 사람이 이 말을 Peter Norvig이 한 것으로 인용하지만, 실제로 이 말을 했는지는 불분명합니다. 비슷한 취지의 슬라이드는 여기에서 찾아볼 수 있습니다:
    - Peter Norvig, 1996, “Design Patterns in Dynamic Languages”
    - https://www.norvig.com/design-patterns/
2. Paul Graham, 2002, “Revenge of the Nerds”,
    - https://paulgraham.com/icad.html
3. Mark Dominus, 2006, “Design patterns of 1972”,
    - https://blog.plover.com/2006/09/11/
4. Allen B. Downy, A Little Book on Semaphore
5. Jonathan Aldrich, 2020, “OO history: Simula and Smalltalk”,
    - https://www.cs.cmu.edu/~aldrich/courses/17-396/slides/oo-history.pdf

당신이 몰랐던 객체지향 Multiple Dispatch

감사합니다