

Real World Cats

@ikhoon

스칼라 여정기

리치자바 시절 🐦 - 스칼라 별거아니네, 람다 편하네 건방젊 (~ 2015)

모나드 겨우 알아가던 시절 😂 - 이런것도 있어? 신났음 (2016)

Category theory 공부하던 시절 😡 - 좌절... 끝이 어디냐! (2017)

Pure functional로 가는길 🚂 - 새로운 시작 (2018 ~)

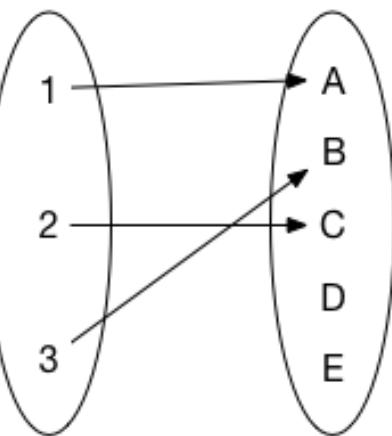
인생무상이로다...

기본 부터 다시 시작해보자, $y = f(x)$

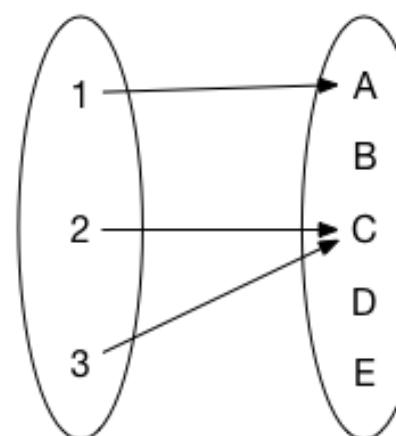
1. 다음중 함수가 아닌것은?

1. 다음중 함수가 아닌것은? [2점]

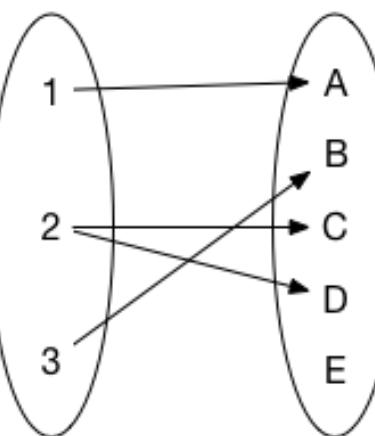
①



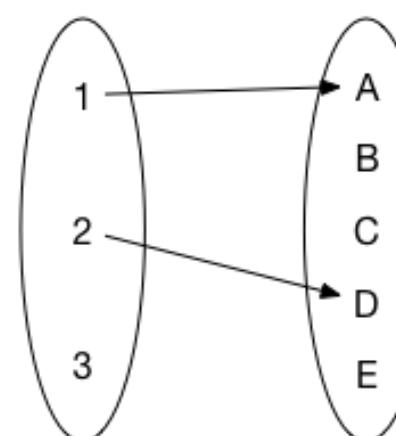
②



③



④



기본 부터 다시 시작해보자, $y = f(x)$

1. 다음중 함수가 아닌것은?

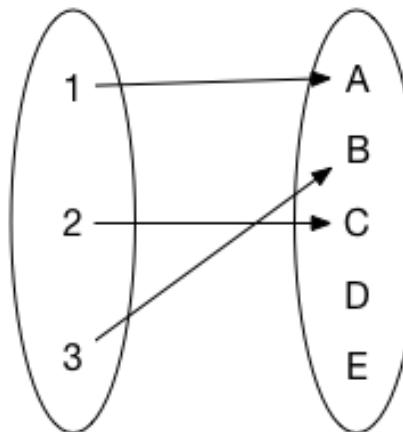
정답 : 3, 4

"함수가 아니지만 함수라는 이름이 붙은
부분 정의 함수의 개념이 존재한다."²⁹

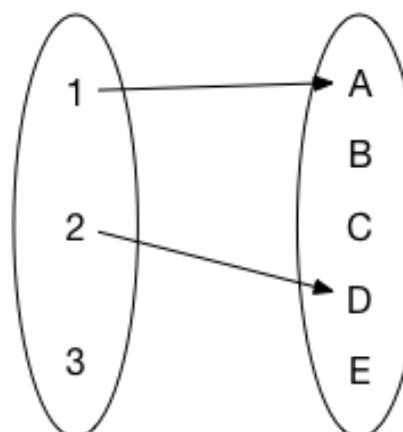
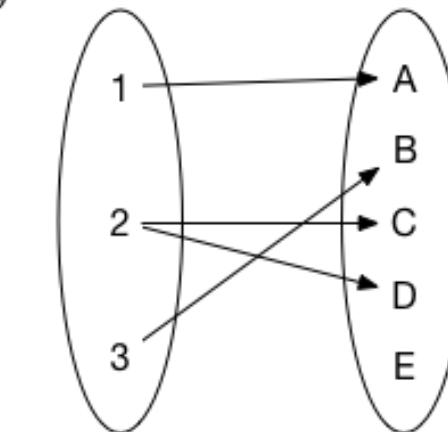
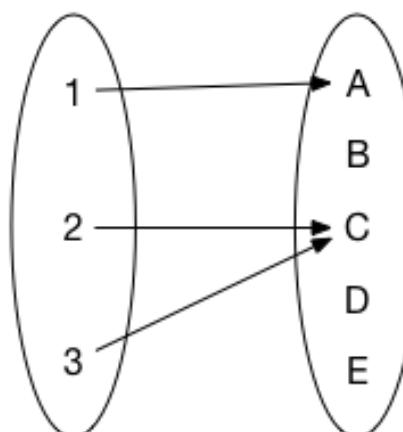
²⁹ https://ko.wikipedia.org/wiki/%ED%95%A8%EC%88%98%EB%B6%80%EB%B6%84_%EC%A0%95%EC%9D%98_%ED%95%A8%EC%88%98_%C2%B7_%EB%8B%A4%EA%B0%80_%ED%95%A8%EC%88%98

1. 다음중 함수가 아닌것은? [2점]

①



②



기본 부터 다시 시작해보자, $y = f(x)$

그러면 우리가 매일 함수라고 호칭하며 만드는 함수는?

2. 다음중 함수가 아닌것은?

2. 다음중 함수가 아닌것은? [2점]

①

```
def factorial(n: Int): BigInt = {  
    require(n < 0, "n은 0보다 큰 양수이어야 합니다")  
    if (n == 0) 1  
    else n * factorial(n - 1)  
}
```

②

```
def factorial(n: Int): Either[String, BigInt] = {  
    if (n < 0) Left("n은 0보다 큰 양수이어야 합니다")  
    else if (n == 0) Right(1)  
    else factorial(n - 1).map(_ * n)  
}
```

아 내가 만든건 함수가 아니였구나... 함수가 뭐길래

2. 다음중 함수가 아닌것은? [2점]

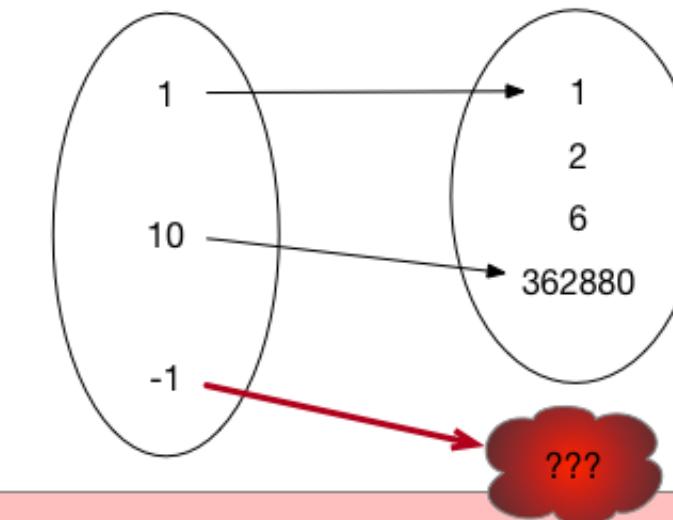


```
def factorial(n: Int): BigInt = {
    require(n < 0, "n은 0보다 큰 양수이어야 합니다")
    if (n == 0) 1
    else n * factorial(n - 1)
}
```

```
scala> factorial(0)
res1: BigInt = 1

scala> factorial(10)
res2: BigInt = 3628800
```

```
scala> factorial(-1)
java.lang.IllegalArgumentException: requirement failed: n은 0 혹은 그보다 큰 양수이어야 합니다
at scala.Predef$.require(Predef.scala:277)
at .factorial(<console>:12)
... 36 elided
```



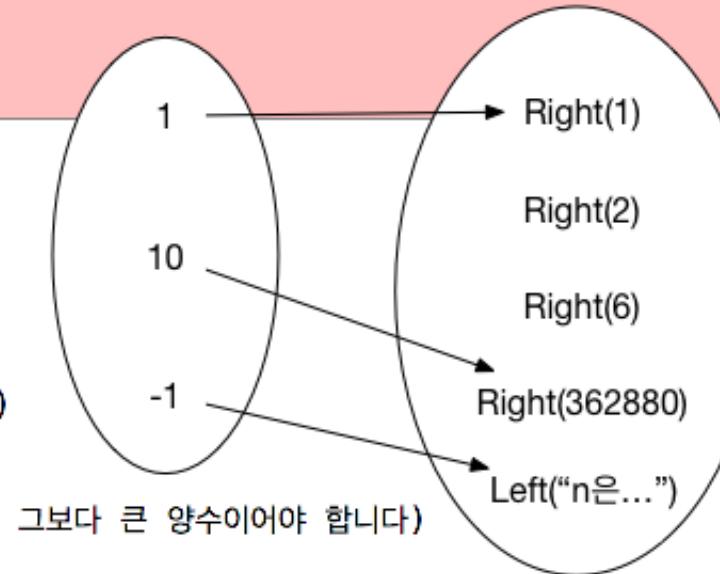
②

```
def factorial(n: Int): Either[String, BigInt] = {
    if (n < 0) Left("n은 0보다 큰 양수이어야 합니다")
    else if (n == 0) Right(1)
    else factorial(n - 1).map(_ * n)
}
```

```
scala> factorial(0)
res4: Either[String,BigInt] = Right(1)

scala> factorial(10)
res5: Either[String,BigInt] = Right(3628800)

scala> factorial(-1)
res6: Either[String,BigInt] = Left(n은 0 혹은 그보다 큰 양수이어야 합니다)
```



```
def factorial(n: Int): Either[String, BigInt] =  
  if (n < 0) Left("n은 0 혹은 그보다 큰 양수이어야 합니다")  
  else if (n == 0) Right(1)  
  else factorial(n - 1).map(_ * n)
```

진짜 함수를 만들기 위해서 cats를 사용할때입니다.
Either와 같은 Monadic 타입 다루는데 cats만한게 없으니까요 ;-)

수학에서는 함수라 불리고

코딩할때는 순수함수라 불리는 녀석과 함께

순수함수 + 프로그래밍에 대해 알아보자

Functional Programming

순수 함수의 정의 및 장점²

$$\text{PF} = \text{ODI} + \text{NSE}$$

Pure Function	Output Depends on Input	No Side Effects
---------------	-------------------------	-----------------

- They're easier to reason about
- They're easier to combine
- They're easier to test
- They're easier to debug
- They're easier to parallelize
- They are idempotent
- They offer referential transparency
- They are memoizable
- They can be lazy

² Book, Functional Programming Simplified - Alvin Alexander

스칼라에서 순수 함수의 장점 활용하기

- 순수 함수의 장점을 살려놓은 라이브러리들이 많이 있다.

합성(**compose**), 병렬처리(**parallel**)

저장(**memoize**), 느긋함(**lazy**)를 잘 지원해준다.

- DRY(Don't Repeat Yourself)

직접 만들려하지 말자

어떤 라이브러리를 사용할것인가?

조건 1 - 문서화가 잘되어 있고 참조 문서가 많은가?

The screenshot shows a dark-themed web page for the Cats library. On the left, there's a sidebar with a red hexagonal logo containing a white cat icon, followed by the word "Cats". Below the logo are several navigation links: "Type Classes", "Semigroups and Monoids", "Semigroup", "Monoid", "Applicative and Traversable", "Functors", and "Folds". The main content area has a white background and a dark header "Type classes". The text explains that type classes are a powerful tool for ad-hoc polymorphism, often known as overloading. It notes that while many polymorphic code uses type parameters like Java generics, functional programming often combines type classes with parametric polymorphism. A section titled "Example: collapsing a list" shows Scala code snippets for summing integers, concatenating strings, and unioning sets using foldRight.

```
def sumInts(list: List[Int]): Int = list.foldRight(0)(_ + _)

def concatStrings(list: List[String]): String = list.foldRight("")(_ ++ _)

def unionSets[A](list: List[Set[A]]): Set[A] = list.foldRight(Set.empty[A])(_ union _)
```

- 잘정리된 공식 문서
- Scala with Cats¹⁵ 책도 있음
- 많은 레퍼런스 문서, 구글에서 "scala cats"로 검색시 458,000개 결과

¹⁵ <https://underscore.io/training/courses/advanced-scala/>

어떤 라이브러리를 사용할 것인가?

조건 2 - 안정적이고 활발하고 지속적으로 개발되고 있는가?



- 209명의 컨트리뷰터
- 3452개의 커밋
- 전체 Pull Request 1524개(42개 Open, 1482개 Closed)
- 1.0.0 릴리즈 이후 바이너리 호환성을 중요시 하고 있다.

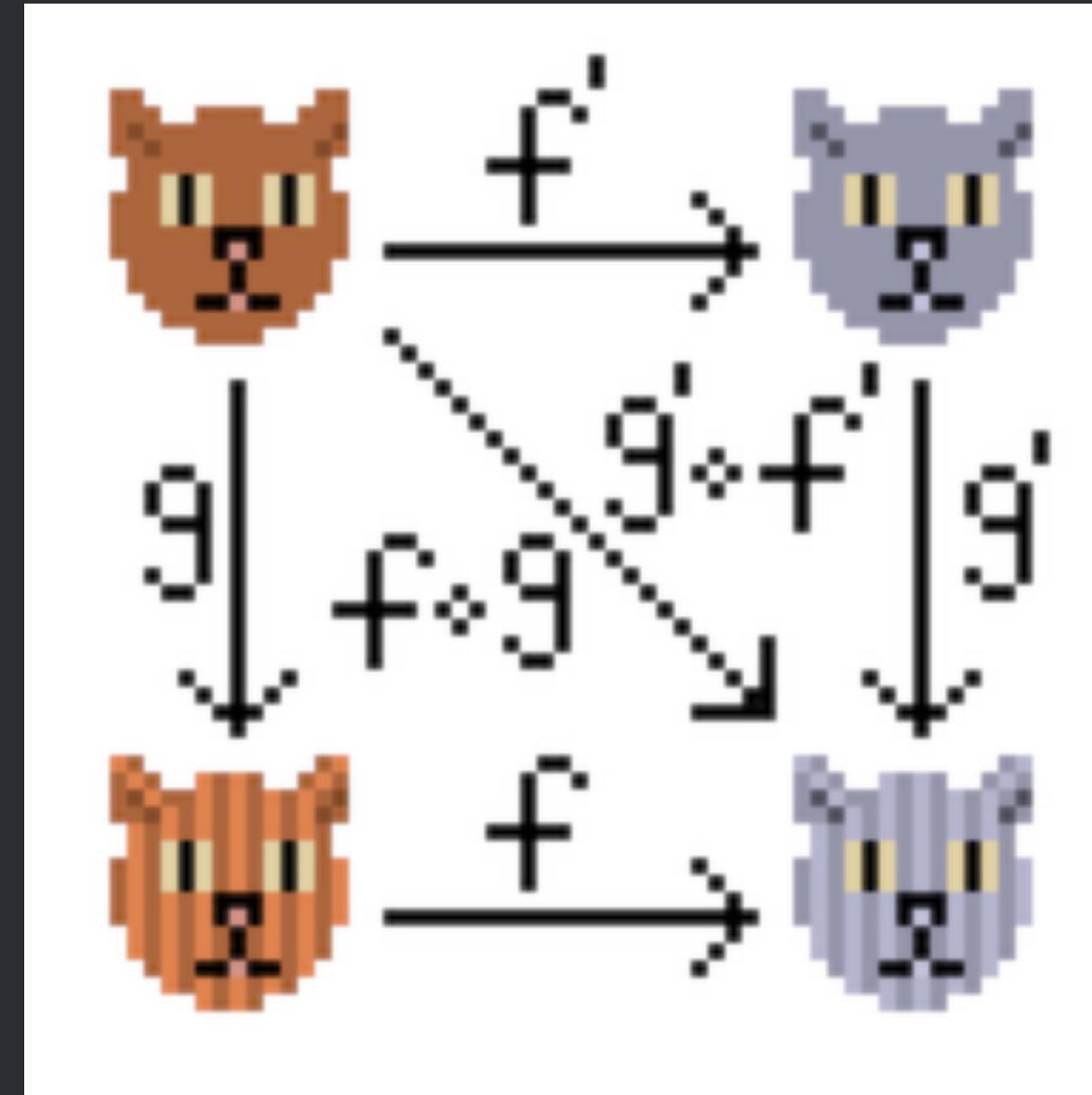
어떤 라이브러리를 사용할것인가?

조건 3 - 라이브러리와 관련된 ecosystem이 잘 구축되어 있는가?

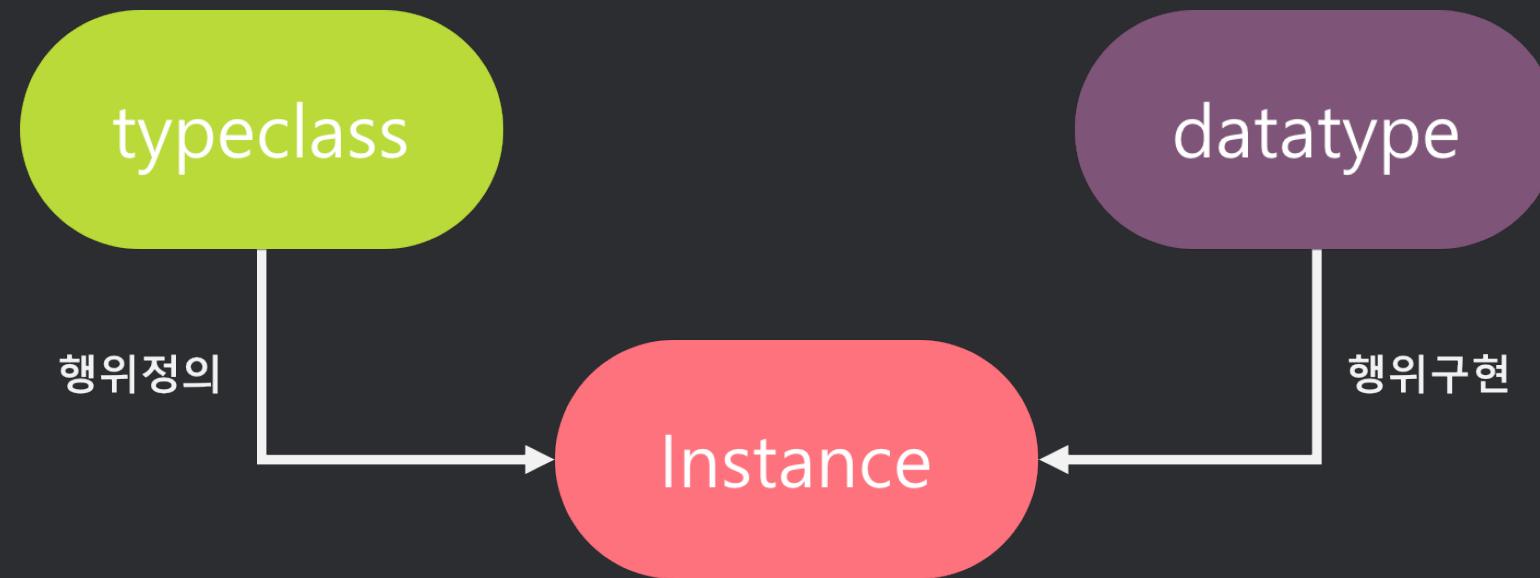
- **circe**: pure functional JSON library
- **doobie**: a pure functional JDBC layer for Scala
- **finch**: Scala combinator library for building Finagle HTTP services
- **FS2**: compositional, streaming I/O library
- **http4s**: A minimal, idiomatic Scala interface for HTTP
- **Monix**: high-performance library for composing asynchronous and event-based programs

다양한 목적을 가진 라이브러리들에서 사용되고 있으며 홈페이지에 언급된 라이브러리만 **31개**이다.

Cats - 믿을만 오픈소스 라이브러리이다.



Cats의 구성



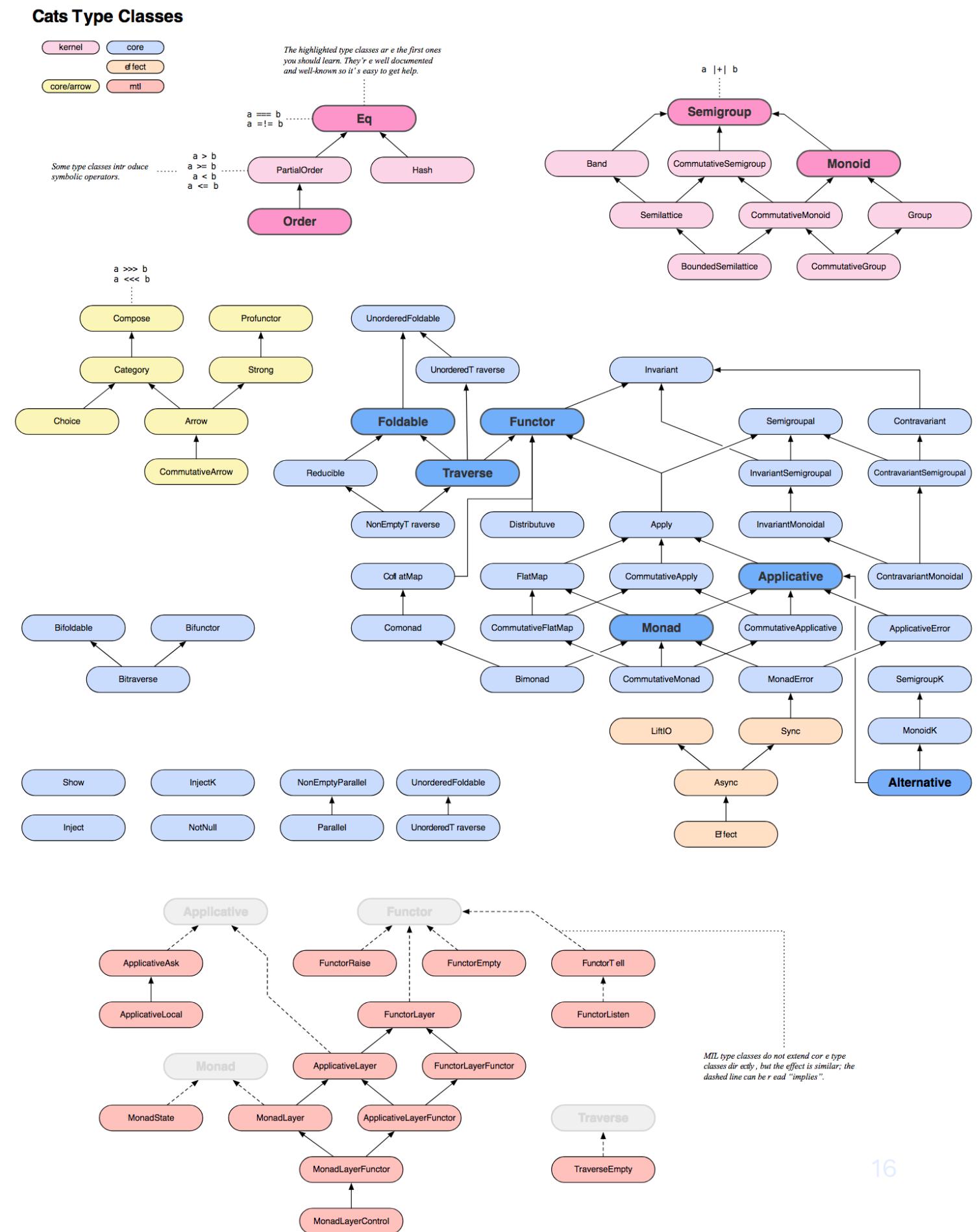
- Typeclass는 행위를 정의하는 인터페이스의 한 형태이다.¹⁷
- Datatype이 '특정 A' typeclass의 instance가 있다는 의미는 '특정 A' typeclass의 행위를 구현하고 지원한다는 뜻이 된다.
- ex) Option(datatype)은 Monad(typeclass)의 행위를 구현(instance)했기 때문에 모나드의 인터페이스(pure, flatMap)을 사용할수 있다.

¹⁷ <http://learnyouahaskell.com/types-and-typeclasses>

Cats의 구성

- Type classes ³⁰
 - Data types
 - Instance

³⁰ <https://github.com/tpolecat/cats-infographic>



Cats의 구성

- Type classes
 - Data types
 - Instance

많고 복잡하다.

하지만 Core 기능만 알자.
그거면 충분하다.

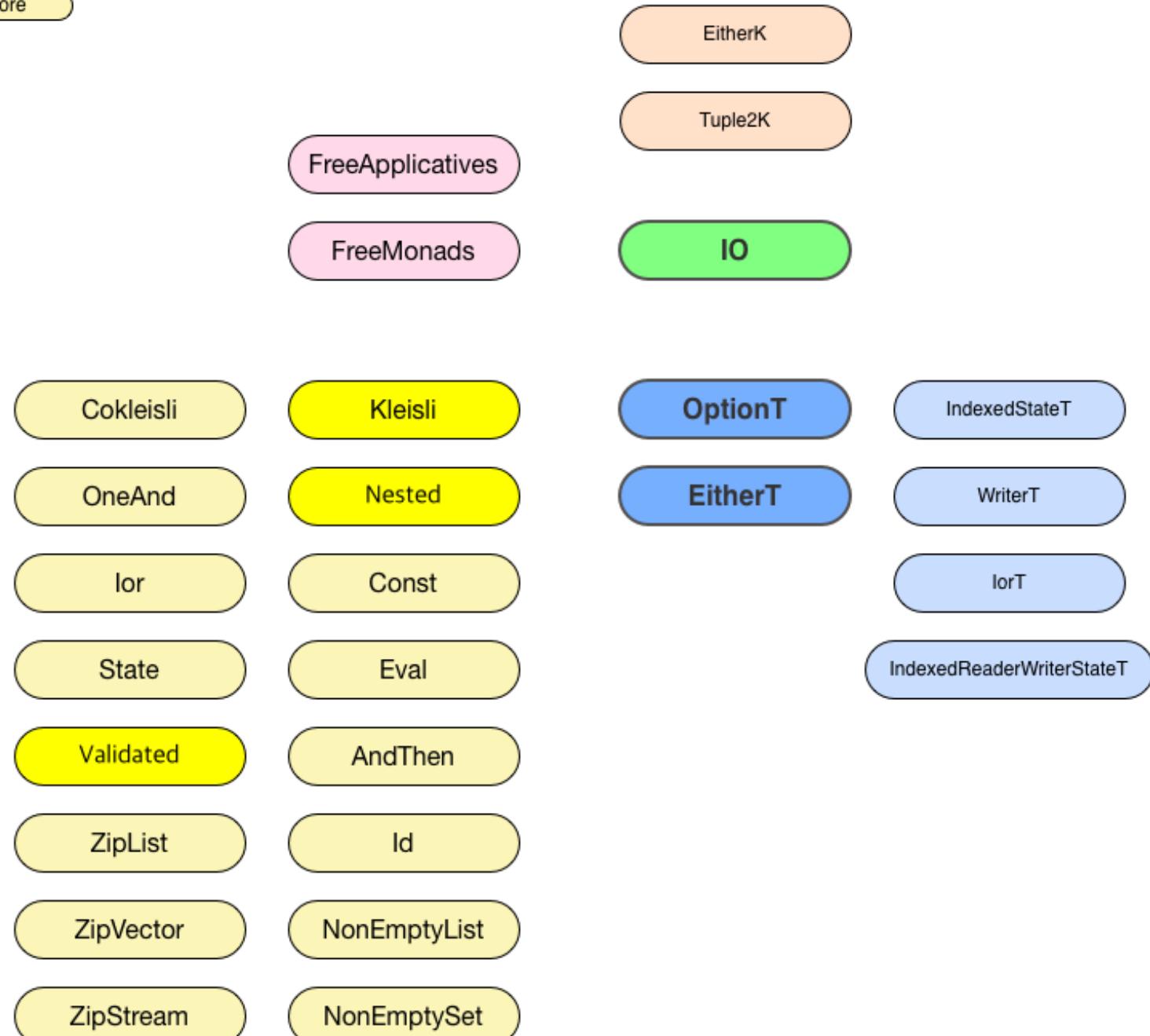


Cats의 구성

- Type classes
- Data types
- Instance

Cats Data Types

free mtl
effect higher kind
core

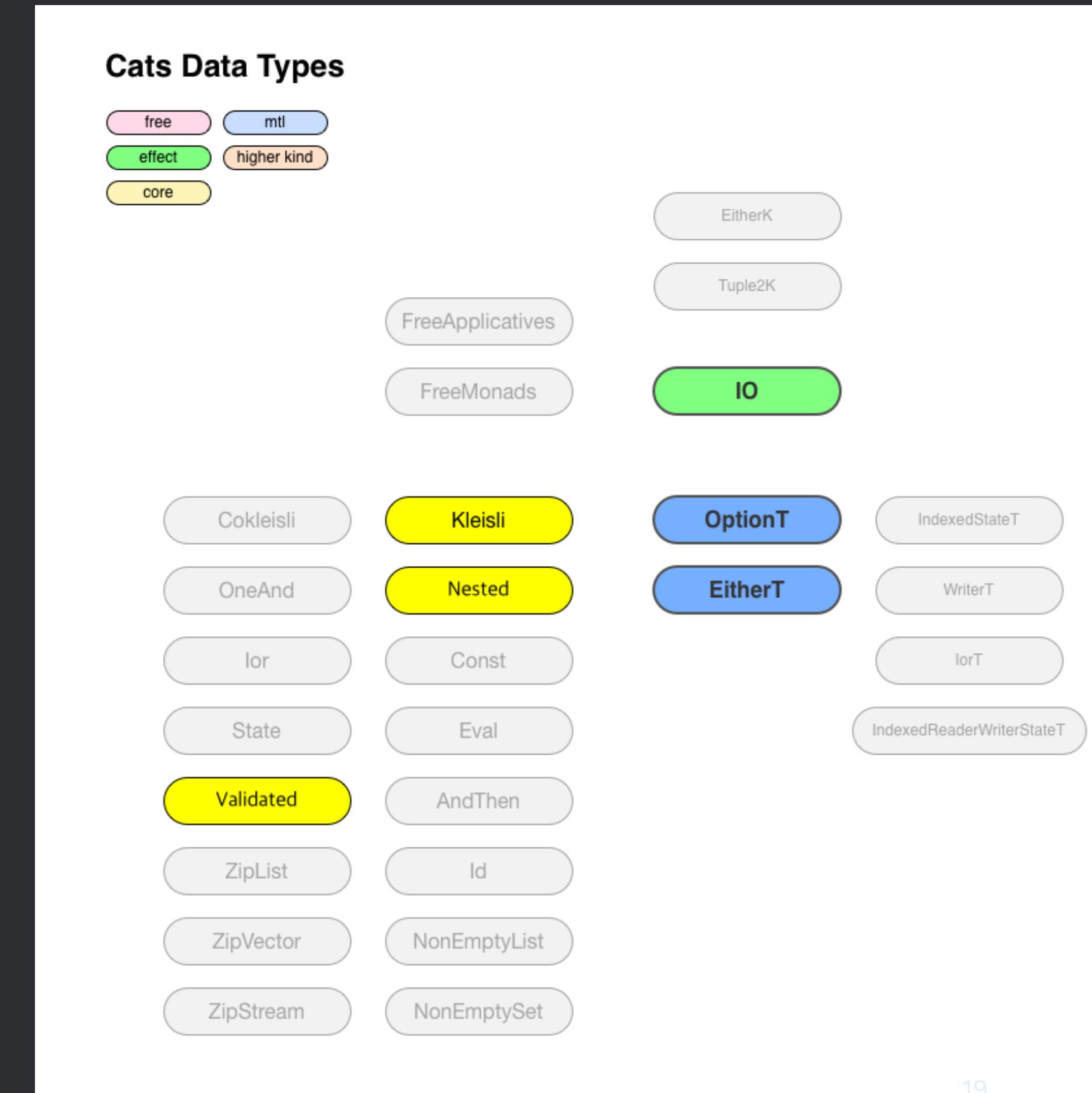


Cats의 구성

- Type classes
- Data types
- Instance

cats에서 제공해주는 자료구조형이다.

서로간에 합성을 할수 있게 도와주고
특히 OptionT와 EitherT는 👍👍이다.



Cats의 구성

- Type classes
- Data types
- **Instance**

Instance는 Datatype과 typeclass를 연결시켜주는 고리 역할이다.
Data type과 type class 사이는 구현이 불가능한 관계인 경우도 있다.

하지만 아직은 신경쓰지말자.

기본적인 core datatype과 core typeclass의 instance는 다 구현되어 있다.¹⁸

¹⁸ <https://typelevel.org/cats/typeclasses.html#incomplete-type-class-instances-in-cats>

Cats 시작하기 전에 - Monad 왜 중요한가?

함수형 언어를 공부하면 Monad에 대한 이야기가 많다.

Monad가 왜 중요할까?

Monad 위키백과 정의 ⁸

C가 범주라고 하자. 그렇다면 자기 함자 $C \rightarrow C$ 들을 대상으로 하고, 이들 사이의 자연 변환들을 사상으로 하는 자기 함자 범주 $\text{End}(C)$ 를 생각하자. $\text{End}(C)$ 는 모노이드 범주이며, 따라서 $\text{End}(C)$ 속의 모노이드 대상을 생각할 수 있다. $\text{End}(C)$ 의 모노이드를 C의 모나드라고 한다.



⁸ [https://ko.wikipedia.org/wiki/%EB%AA%A8%EB%82%98%EB%93%9C_\(%EB%B2%94%EC%A3%BC%EB%A1%A0\)](https://ko.wikipedia.org/wiki/%EB%AA%A8%EB%82%98%EB%93%9C_(%EB%B2%94%EC%A3%BC%EB%A1%A0))

Cats 시작하기 전에 - Monad 왜 중요한가?

함수형 언어를 공부하면 Monad에 대한 이야기가 많다.

모나드가 왜 중요할까?

수학이 아닌

소프트웨어 설계의 관점에서의 flatMap(Monad)에서 알아보자.

Cats 시작하기 전에 - 기본이 되는 함수

Ordered를 상속받아 compare 함수를 구현하면
4개의 비교함수(<, <=, >, >=)를 공짜로 얻을 수 있다.

```
case class Version(major: Int, minor: Int, patch: Int) extends Ordered[Version] {  
    def compare(that: Version): Int =  
        if(major > that.major) 1  
        else if (major == that.major && minor > that.minor) 1 ...  
        else -1  
}
```

```
Version(1, 11, 1) < Version(1, 1, 1) // false  
Version(1, 10, 1) > Version(0, 0, 1) // true  
Version(10, 9, 3) <= Version(0, 0, 1) // false
```

**소프트웨어 설계의 관점에서의 Monad는
코드의 재사용성을 극대화 할수 있는 도구이다.**

기본이 되는 두개의 함수 pure와 flatMap만 구현하면 많은 유용한 함수를 공짜 로 얻을수 있다.(약 110개)

- map = pure + flatMap
- ap = flatMap + map
- product = map + ap
- map2 = map + product

소프트웨어 설계의 관점에서의 Monad는 코드의 재사용성을 극대화 할 수 있는 도구이다.

```
def map[A, B](fa: F[A])(f: A => B): F[B] =  
  flatMap(fa)(f andThen pure)
```

```
def ap[A, B](fa: F[A])(ff: F[A => B]): F[B] =  
  flatMap(fa)(a => map(ff)(f => f(a)))
```

```
def product[A, B](fa: F[A], fb: F[B]): F[(A, B)] =  
  ap(fb)(map(fa)(a => (b: B) => (a, b)))
```

```
def map2[A, B, Z](fa: F[A], fb: F[B])(f: (A, B) => Z): F[Z] =  
  map(product(fa, fb))(f.tupled)
```

어려워 말아요 **cats** 쓰다보면 조금씩 이해되더라... 🤝

카테고리 이론이 뭔지... 모나드가 뭔지 😕

복잡한 개념은 잠시 내려놓고

기본 개념과 사용 예를 중심으로 ➡

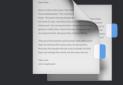
"부족한 부분은 추가로 backup slide에 자료를 남겨 놓겠습니다"

그럼 💻 실무에서는 언제 어떻게 적용할것인가?

1. 많은 양의 데이터 연동 처리
2. 비동기 필터링
3. 확장가능한 cache 전략
4. nullable 데이터 효과적으로 다루기

1. 많은 양의 데이터 연동 처리

오늘의 업무



100개의 상품 정보를 가져와서 처리후 화면에 보여줘야함.

REST API기준으로 API를 설계하다 보면

한꺼번에 100개의 상품을 조회 API가 없는 경우도 많다... 🤯

GET /v1/api/items/:id

기존 API를 활용 100개 상품 조회를 구현해보자.

```
val itemIds: List[Int]
// 단건의 상품 정보를 가져오는 API
def findItemById(itemId: Int): Future[Option[Item]]
```

느린 API 응답속도

Problem

하나의 API가 100ms 걸린다면?
100개의 상품 정보를 가져오는데
합쳐서 10초?! 이건 쓸수 없다. 🤦

다시 API담당자에게
똑똑 바쁘시겠지만 죄송하지만...
업무가 바쁘셔서 2주 기다려 달란다. 😭

기획자가 쪼은다
이번주까지 해달라는데 😱

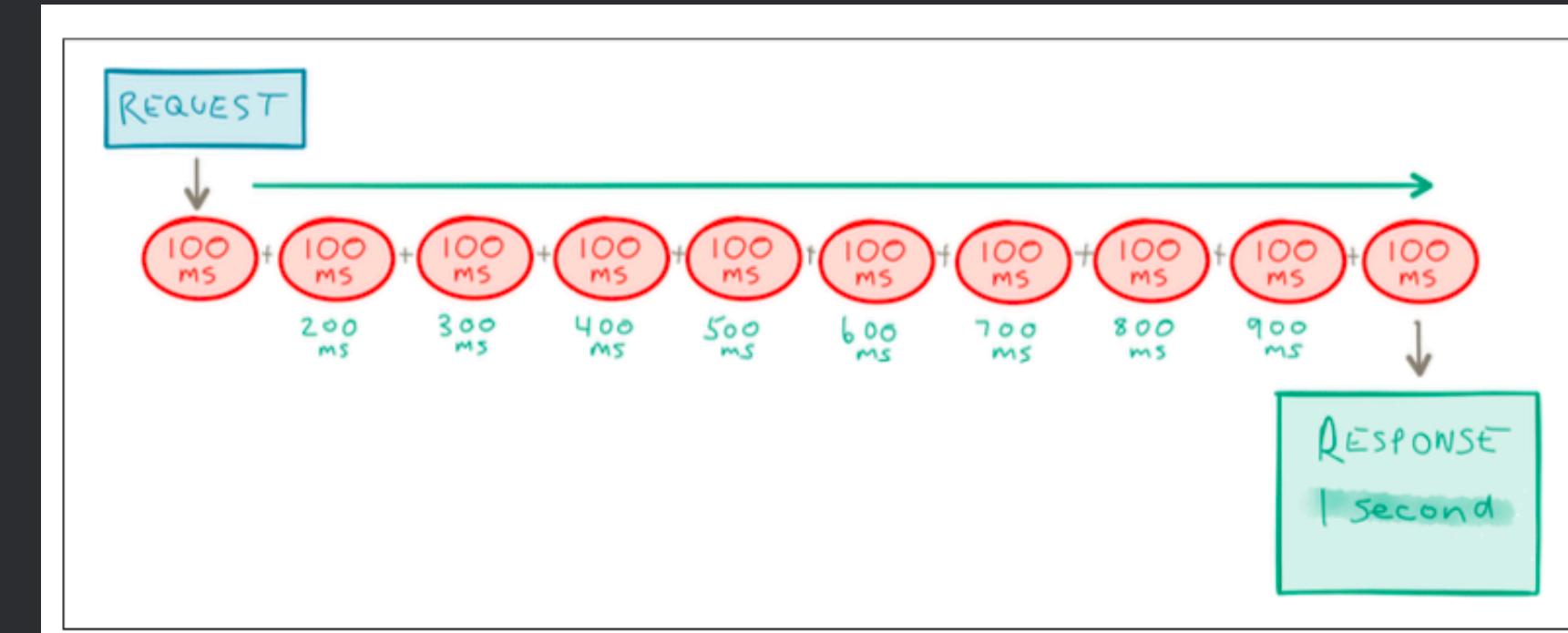


Figure 2-5. Synchronous requests increase latency

병렬 처리를 통한 응답속도 향상

Solution

병렬로 처리하자²⁵

하지만 동시에 100개를 요청을 상대방 서버로 할수 없다.

100개를 동시에 요청하면

상대방 서버에서 DDOS 공격처럼 느껴질수 있다.

부담이 안갈 수준으로 병렬로 호출하자.

100개를 한번에 10개씩 10번에 걸쳐 처리하면

부하에 대한 부담도 없고 빠르게 처리할수 있다.

예상되는 처리 시간은 1초이다.😊

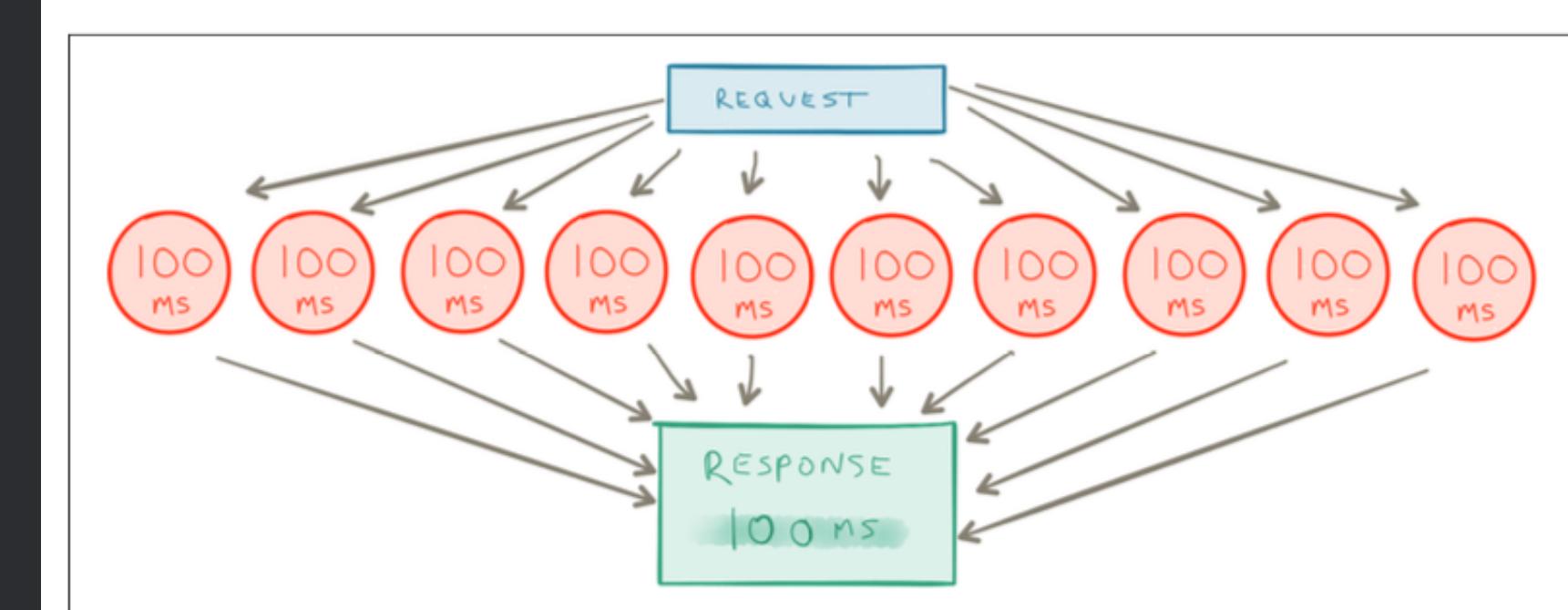


Figure 2-6. Asynchronous requests execute as fast as the slowest request

²⁵ Reactive Microservices Architecture By Jonas Bonér,
Co-Founder & CTO Lightbend, Inc.

순차와 병렬 처리의 조합

flatMap

데이터를 순차적으로 처리할수 있다.

traverse

데이터를 동시에 처리(map)하고 처리된 결과를 모아(reduce)준다.

flatMap과 traverse를 조합해서 비지니스의 요구사항을 구현할수 있다.

```
def flatMap[A, B](fa: F[A])(f: A => F[B]): F[B]
```

```
def traverse[G[_]: Applicative, A, B](fa: F[A])(f: A => G[B]): G[F[B]]
```

Future, Lazy, Parallel 그리고 traverse특성에 대해 조금 더 알아보기

— Traverse

Traverse 자체는 병렬 처리를 위한 typeclass가 아니다.

그렇기 때문에 traverse 함수 자체로는 병렬 동작을 보장할 수 없다.

— Parallel (parTraverse, parMapN)

cats에서는 병렬처리를 위한 typeclass가 별도로 존재한다.

하지만 Scala Future는 Parallel의 instance가 없다.²⁸

— Scala Future (Eager)

대신 Eager evalution되어 traverse 함수가 병렬로 동작을 한다.

— Monix Task (Lazy)

Lazy evaluation되고 traverse 함수가 순차적으로 실행된다.

monix의 Task는 parTraverse를 써야지만 병렬로 동작한다.

코드의 동작 측면에서는 Monix의 Task가 더 예측가능하고 좋다.

```
def findItemId(itemId: Int): Task[Item] // Monix의 Task
itemIds.traverse(findItemId)      // 순차 진행하며 10초 걸림
itemIds.parTraverse(findItemId)   // 병렬 처리되며 1초 걸림
```

²⁸ https://www.reddit.com/r-scala/comments/7qimqg/cats_parallel_future_traverse/

한번에 10개씩, 10번에 나누어 처리하자

사용할 API들

- grouped : 10개의 단위로 묶고
- foldLeft : List의 요소에 순차 접근하여
- flatMap : 10개의 처리가 완료된 후에 다음 10개가 처리되게 하며
- traverse : 10개를 동시에 처리할수 있도록 한다.(혹은 parTraverse)

구현은 생각보다 간단함 😊

```
val items = itemIds
  .grouped(10).toList                                // 그룹
  .foldLeft(List.empty[Item].pure[Future]) ((acc, group) =>
    acc.flatMap(xs =>                                // 순차
      group.traverse(findItemById).map(xs :::_))) // 병렬
```

두번째 이야기

1. 많은 양의 데이터 연동 처리
2. 비동기 필터링

2. 비동기 필터링

Problem

주어진 상품 ID들 중에 아직 유효한 상품ID만 필터링하고 싶다.
그런데 상품의 상태는 비동기로 가져와야 한다.

```
val itemIds: List[Int]
// 상품 상태를 가져오는 API
def isActiveItem[F[_]](itemId: Int)(implicit F: Effect[F]): F[Boolean]
```

필요한 로직은 filter + async

2. 비동기 필터링 구현하기

원초적인(생각나는데로) 문제를 풀어본다.

// 1. map을 이용해 상품 정보를 읽어온다.

```
val step1: List[IO[(Int, Boolean)]] = itemIds.map(itemId =>
  isActiveItem[IO](itemId).tupleLeft(itemId))
```

// 2. IO가 List안에 있으니 밖으로 꺼내자

```
val step2: IO[List[(Int, Boolean)]] = step1.sequence
```

// 3. 이제 겨우 filter해본다.

```
val step3: IO[List[(Int, Boolean)]] = step2.map(xs => xs.filter(_._2))
```

// 4. 필요없는 Boolean은 제거

```
val step4: IO[List[Int]] = step3.map(_.map(_._1))
```

비동기 필터링 구현하기

구현하기 불편하고 복잡하다. 

나는 필터링만 하고 싶다.

더 나은 방법이 없을까?

비동기 필터링 구현하기

구현하기 불편하고 복잡하다. 🤢

나는 필터링만 하고 싶다.

더 나은 방법이 없을까?

있다! 

filterA를 활용하자.

filterA 함수를 이용한 비동기 필터링 구현하기

구현이라 하기도 애매하다. 너무 짧다. filterA함수 하나만 호출하면 된다.

```
import cats.mtl.implicits._

// filterA함수를 통해서 앞에 했던 로직을 표현할수 있다.
val activeItem: IO[List[Int]] = itemIds.filterA(isActiveItem[IO])
```

앞서 4개의 step을 통해 했던 작업은 filterA 함수하나면 된다.

훨씬 코드가 간결해졌다. 😎 FP 만세!!!

외부에서 데이터의 상태의 정보를 가져와 연동하는 곳에서는 유용한 함수라 생각된다.

filterA 함수의 구조 알아보기

```
def filterA[G[_], A]
  (fa: F[A])(f: A => G[Boolean])
  (implicit G: Applicative[G]): G[F[A]]
```

- filterA의 predicate
함수가 $A \Rightarrow G[\text{Boolean}]$ 형태이다.
- G는 Applicative
Applicative의 instance를 가 있는 List, Future, Option가 올수 있다.
- filterA의 활용
꼭 비동기에 국한된건 아니다.
 $f: A \Rightarrow \text{Option}[\text{Boolean}]$ 등 상황에 맞는 함수를 조합해서 사용할수 있다.

cats 1.x 에서 filterA 사용하기

cats 0.x 버전에서는 filterA는 import만 해도 사용할수 있고

```
import cats.implicitly._ // 0.x 전에는 이것만
```

cats 1.x로 넘어가면서 filterA는 **cats-mtl**²³에 이동되었다.
그래서 사용을 하려면 의존성 추가와 import를 해줘야 한다.

```
libraryDependencies += "org.typelevel" %% "cats-mtl" % "0.2.1"  
import cats.mtl.implicitly._ // 추가로 import해줘야함
```

²³ <https://github.com/typelevel/cats-mtl/blob/f3bd9aa4c8c0466d8cb4aa25bfcc6480531929c5/core/src/main/scala/cats/mtl/TraverseEmpty.scala#L48>

세번째 이야기

1. 많은 양의 데이터 연동 처리
2. 비동기 필터링
3. 확장가능한 cache 서비스 구현

3. 확장가능한 cache 서비스 구현

우리는 빠른 응답을 위해 cache에 결과값을 저장해서 사용하는 경우가 많다.
Cache miss가 발생을 보안하기 위해 backup에서 데이터를 읽어와서 처리를 구현했다.

Problem - 아래 코드 자체가 문제이다.
무엇을 말하고자 하는지 한눈에 들어 오지 않는다.
많은 패턴 매칭은 코드의 가독성을 떨어 뜨린다. 중복코드가 존재한다. 
cache의 layer가 늘어나거나 줄어들어 코드를 바꾸기엔 확장성도 떨어진다.

```
def fetch[F[_]](id: Int)(implicit F: Effect[F]): F[Option[User]] =  
  fetchLocalCache(id).flatMap {  
    case cached @ Some(_) => F.pure(cached) //local cache에서 데이터를 가져오고  
    case None => //그 값이 있으면 사용한다  
      fetchRemoteCache(id).flatMap {  
        case cached @ Some(_) => F.pure(cached) //remote cache에서 데이터를 가져오고  
        case None => fetchDB(id) //그 값이 없으면 사용한다.  
      }  
    } //또 없으면 DB에서 가져온다.
```

Problem - 아래 코드 자체가 문제이다.

```
def fetch[F[_]](id: Int)(implicit F: Effect[F]): F[Option[User]] =  
  fetchLocalCache(id).flatMap { // local cache에서 가져오고  
    case cached @ Some(_) => F.pure(cached) // 그 값이 있으면 사용한다  
    case None => // 없으면  
      fetchRemoteCache(id).flatMap { // remote cache에서 가져오고  
        case cached @ Some(_) => F.pure(cached) // 그 값이 있으면 사용한다.  
        case None => fetchDB(id) // 또 없으면 DB에서 가져온다.  
      }  
  }
```

Cache miss 처리 구현의 문제점

패턴 매칭을 없애고
각각의 cache layer간의 커플링을 줄이고
이를 유지보수 하기 쉽게 정형(pattern)화 할수 있을까?

Cache miss 처리 구현의 문제점

패턴 매칭을 없애고
각각의 cache layer간의 커플링을 줄이고
이를 유지보수 하기 쉽게 정형(pattern)화 할수 있을까?

있다! 

SemigroupK의

“pick the first winner”¹⁰ 원칙을 적용하면 된다.

¹⁰ Book, Functional Programming for Mortals with Scalaz - Sam Halliday

SemigroupK 구조 및 특징 알아보기

구조 - 인터페이스는 단순하다. 두개의 $F[A]$ 타입을 합친다

```
@typeclass trait SemigroupK[F[_]] {  
    @op("<+>") def combineK[A](x: F[A], y: F[A]): F[A]  
}
```

특징 - Plus (or as cats have renamed it, SemigroupK) has "pick the first winner, ignore errors" semantics. Although it is not expressed in its laws.¹¹

¹¹ <https://github.com/typelevel/cats-effect/issues/94#issuecomment-351736393>

SemigroupK 예제를 통해 알아보기 - IO

IO 자료구조에 대해서는 첫번째 값만 취하려 한다.

만약 첫번째에서 에러가 발생하면 무시하고 두번째 연산을 진행한다.

// "First"만 실행이 된다. IO.apply는 lazy 연산을 하기 때문이다.

```
val res1 = IO { "First" } <+> IO { "Second" }
res1.unsafeRunSync() // res1: "First"
```

// "First Error" 에러💥는 무시되고 두번째 값 "Second"가 나온다.

```
val res2 = IO.raiseError(new Exception("First Error")) <+> IO { "Second" }
res2.unsafeRunSync() // res2: "Second"
```

SemigroupK 예제를 통해 알아보기 - Option

Option 자료 구조에 대해서는 || 연산과 같은 동작이다.

더해지는 연산이 아니다.

앞쪽에 Some 타입이면 뒷부분 연산은 진행하지 않는다.

```
1. some <+> none    // Some(1)   
1. some <+> 2. some // Some(1)   
none   <+> 2. some // Some(2) 
```

SemigroupK를 이용한 Cache miss 처리하기

이제 이코드는

```
def fetch[F[_]](id: Int)(implicit F: Effect[F]): F[Option[User]] =  
  fetchLocalCache(id).flatMap {  
    case cached @ Some(_) => F.pure(cached) //local cache에서 데이터를 가져오고  
    case None => //그 값이 있으면 사용한다  
      fetchRemoteCache(id).flatMap {  
        case cached @ Some(_) => F.pure(cached) //remote cache에서 데이터를 가져오고  
        case None => fetchDB(id) //그 값이 있으면 사용한다.  
      } //또 없으면 DB에서 가져온다.  
  }  
}
```

SemigroupK를 이용한 Cache miss 처리하기

훨씬 코드가 간결해졌다. 😎 FP 만세!!!

```
def fetch[F[_]: Effect](id: Int): F[Option[User]] = (
    OptionT(fetchLocalCache(id)) <+>
    OptionT(fetchRemoteCache(id)) <+>
    OptionT(fetchDB(id)))
).value
```

각각의 cache layer간의 코드가 커플링이 없어지고
쉽게 cache layer를 늘리거나 줄일수 있다.

마지막 이야기

1. 많은 양의 데이터 연동 처리
2. 비동기 필터링
3. 확장가능한 cache 전략
4. nullable 데이터 효과적으로 다루기

4. nullable 데이터 효과적으로 다루기

Problem

많은 비동기 API의 반환되는 데이터가 null이 될수 있어서
함수의 리턴타입이 Future[Option[A]]가 된다.

- DB에서 select one
- Cache에서 데이터를 get
- Http request에 대한 응답

Future 하나도 처리하기 힘든데 Option까지...
두개의 다른 monadic이 합쳐졌다.. 어떻게 하지? 😱

4. 중첩된 모나드 Future[Option[A]]의 문제점

order -> user -> address를 얻는 로직이 있다 하자.

Future[Option[A]]는 **합성할 수 없기** 때문에 for문 안에 지저분한 패턴 매칭이 난무한다. 

```
for {  
    orderOption <- findOrder(orderId)  
    userOption  <- orderOption match {  
        case Some(order) => findUser(order.userId)  
        case None  => Future.successful(None)  
    }  
    address     <- userOption match {  
        case Some(user) => findAddress(user.id)  
        case None  => Future.successful(None)  
    }  
} yield address
```

4. Monad Transformer Save Us

뭔가 아직은 부족한 나의 스칼라 개발 내공

구글링과 책을 읽다보니

Monad Transformer라는 녀석이 존재했다.

기능을 보고 너무 감동 받았다. 😊

cats를 처음 사용하기 시작한 이유도

Monad Transformer를 datatype을 사용하기 위해서다.

그리고 어느날 github에...

```
/**  
 * Created by Liam.M on 2017. 08. 07..
```

```
* 07/08/2017
```

```
*/ Monad transformer 관련된 이상한 공식을 남긴다... 😕19
```

Monad Transformer 찬양론자가 되었습니다...

하지만 Monad Transformer는 개발 생산성 향상에 레알 도움을 준다. 🤘

```
/**
```

```
* 여기있는 공식만 외우면 여러분은 마틴오더스키가 될것입니다.
```

```
*
```

```
*/19 https://github.com/ikhoon/scala-note/blob/master/scala-exercise/src/test/scala/catsnote/MonadTransformerSaveUs.scala
```

```
class MonadTransformerSaveUs extends FunSuite with Matchers
```

Monad Transformer가 왜 필요한가?

Monad는 합성할수 없다.

이유는 두개의 Monad가 합쳐진 일반화된 flatMap함수를 구현할 수 없기 때문이다. 🤦

그래서 Monad Transformer를 이용해 Monad를 합성한것과 똑같은 효과를 얻고자 한다.

Monad 합성에 대한 좀더 자세한 내용은 Scala with Cats¹⁵ 5장을 참조하면 됩니다. 🙏

```
import cats.Monad
import cats.syntax.applicative._ // for pure
import cats.syntax.flatMap._    // for flatMap
import scala.language.higherKinds

// Hypothetical example. This won't actually compile:
def compose[M1[_]: Monad, M2[_]: Monad] = {
  type Composed[A] = M1[M2[A]]

  new Monad[Composed] {
    def pure[A](a: A): Composed[A] =
      a.pure[M2].pure[M1]

    def flatMap[A, B](fa: Composed[A])
      (f: A => Composed[B]): Composed[B] =
      // Problem! How do we write flatMap?
      ???
  }
}
```

It is impossible to write a general definition of flatMap without knowing something about M1 or M2. However, if we do know something about one

¹⁵ <https://underscore.io/training/courses/advanced-scala/>

Monad Transformer 종류 알아보기

WriterT, StateT등 다양한 Monad Transformer가 있지만 자주 쓰이는 OptionT와 EitherT를 보면

중간의 타입을 고정을 한것에 주목하자.(flatMap을 구현하기 위해서)

OptionT = monad transformer for Option⁵

OptionT[F[_], A] is a light wrapper on an F[Option[A]]

EitherT = monad transformer for Either⁶

EitherT[F[_], A, B] is a light wrapper for F[Either[A, B]]

⁵ <https://typelevel.org/cats/datatypes/optiont.html>

⁶ <https://typelevel.org/cats/datatypes/eitherT.html>

심화 과정 - OptionT 구현 알아보기

너무 디테일한 내용은 피하고 싶었으나 궁금해 하실분을 위해 짧게만 😜
flatMap을 위해 $A \Rightarrow F[Option[A]]$ 를 $Option[A] \Rightarrow F[Option[B]]$ 로 변경하는 코드이다.

```
case class OptionT[F[_], A](value: F[Option[A]]) {  
  
  def flatMapF[B](f: A => F[Option[B]])(implicit F: Monad[F]): OptionT[F, B] = {  
  
    /* 문제점 : F.flatMap은 Option[A] => F[Option[B]] 함수가 필요하다.  
     * 그러나 f 함수의 형태는 A => F[Option[B]]이다.  
     * 해결책 : Option의 패턴 매칭을 통해서 새로운 함수 fn을 만들고  
     *          f 함수를 Option[A] => F[Option[B]] 형태로 변형하자. */  
    def fn(fa: Option[A]): F[Option[B]] = fa match {  
      case Some(v) => f(v)  
      case None => F.pure[Option[B]](None)  
    }  
    OptionT(F.flatMap(value)(fn))  
  }  
  
  def flatMap[B](f: A => OptionT[F, B])(implicit F: Monad[F]): OptionT[F, B] =  
    flatMapF(a => f(a).value)  
  }  
}
```

공식 - F[Option[A]] 있다면 OptionT로 감싸자

이제 난잡한 이코드는 OptionT로 감싸면

```
for {  
    orderOption <- findOrder(orderId)  
    userOption <- orderOption match {  
        case Some(order) => findUser(order.userId)  
        case None => Future.successful(None)  
    }  
    addr <- userOption match {  
        case Some(user) => findAddress(user.id)  
        case None => Future.successful(None)  
    }  
} yield addr
```

공식 - F[Option[A]] 있다면 OptionT로 감싸자

훨씬 코드가 간결해졌다. 😁 FP 만세!!!

```
val address = for {
    order <- OptionT(findOrder(orderId))
    user  <- OptionT(findUser(order.userId))
    addr   <- OptionT(findAddress(user.id))
} yield addr
```

실전 - 모든함수가 Future[Option[A]]를 반환하지는 않는다

A, Option[A], Future[A], Future[Option[A]]

이런 다른 타입을 반환하는 함수끼리 서로 합성을 할수 있어야 real world이다.

아래 코드는 실제 컴파일 되지 않는다. 🤦 타입이 맞지 않기 때문이다.

이를 적절히 lift해서 OptionT로 변환하자.

```
def getUserDevice(userId: String): Future[Option[Device]] = {
    val userId = parseUserId(userId)          // Option[Long]
    val user = findUser(userId)                // Future[Option[User]]
    val phoneNumber = getPhoneNumber(user)     // String
    val device = findDevice(phoneNumber)      // Future[Device]
    device
}
```

실전 공식 - 보조함수를 이용해서 결과타입을 OptionT로 맞추자

- 중간에 $A \Rightarrow Future[Option[B]]$ 있다면? 무조건 OptionT 이다!
- 중간에 $A \Rightarrow Option[C]$ 있다면? 무조건 OptionT.fromOption[Future] 이다.
- 중간에 $A \Rightarrow D$ 있다면? 무조건 OptionT.pure[Future, A] 이다!
- 중간에 $A \Rightarrow Future[E]$ 왔다면? 무조건 OptionT.liftF 이다.
- 반환하는 타입이 Future[Option[A]]라면 여기에 맞추어라 무조건!!!!

```
def findDeviceByUserId(userId: String): Future[Option[Device]] = (for {  
    userId <- OptionT.fromOption[Future](parseUserId(userId)) // Option[Long]  
    user   <- OptionT(findUser(userId))                      // Future[Option[User]]  
    phone  <- OptionT.pure[Future](getPhoneNumber(user))       // String  
    device <- OptionT.liftF(findDevice(phone))                // Future[Device]  
} yield device).value
```

들어주셔서 감사합니다.



Learn^{ing} resources

- <https://typelevel.org/cats/>
공식 문서, 잘 정리되어 있음
- <https://www.scala-exercises.org/cats>
브라우저에서 실습보면서 공부할수 있음
- <http://eed3si9n.com/herding-cats>
아주 친절한 블로그
- <https://underscore.io/training/courses/advanced-scala/>
공짜 무료 이북 핵이득
- (광고) 르 스칼르 코딩단 ↗ slack #typedrug 💊

BACKUP SLIDE

부록 슬라이드 - 준비는 했으나 시간 관계상 🙊

Typeclass로 일관된 API 사용하기
+ 그 외 잡다한 이야기

Monoid 위키백과 정의⁷

모노이드(영어: monoid)는 항등원을 갖는, 결합 법칙을 따르는 이항 연산을 갖춘 대수 구조이다. 군의 정의에서 역원의 존재를 생략하거나, 반군의 정의에서 항등원의 존재를 추가하여 얻는다.



⁷ <https://ko.wikipedia.org/wiki/%EB%AA%A8%EB%85%B8%EC%9D%B4%EB%93%9C>

같은 역할 다른 API

map, flatMap

Monad의 instance를 만들수 있는 monadic Option, List, Future 같은 datatype에서는 대부분 map과 flatMap을 지원한다.

pure, handleError

하지만 상속에 의한 강제가 아니기 때문에 다를수도 있다.

- monad 생성자(pure)
- 에러처리(handleError)

와 같은 영역은 각 API마다 제각각이다.

특히 자바의 라이브러리를 사용한다면 더 다를것이다.

같은 역할 다른 API

data type	성공 생성자	실패 생성자	에러 처리
 scala future	successful	failed	recover
 twitter future	value	exception	handle
 monix task	pure	onErrorRecover	raiseError

너무나도 다르다.

다른 라이브러리로의 변경이 필요하다면 고쳐야 할 부분이 너무 많다.

가장 중요한건 다른 API를 알아야 한다.

난 암기 과목이 싫다 

다른 API 직접 활용하기

EXAMPLE?

Typeclass

Scala Future

Applicative
pure

Functor
map

Monad
flatMap

ApplicativeError
handleError

```
def getItem(id: Long): ScalaFuture[Either[Throwable, ItemDto]] =  
  if(id < 0) ScalaFuture.successful(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .recover { case ex: Throwable => Left(ex)}
```

Monix Task

Twitter Future

```
def getItem(id: Long): MonixTask[Either[Throwable, ItemDto]] =  
  if(id < 0) MonixTask.pure(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .onErrorRecover { case ex: Throwable => Left(ex) }
```

```
def getItem(id: Long): TwitterFuture[Either[Throwable, ItemDto]] =  
  if(id < 0) TwitterFuture.value(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .handle { case ex: Throwable => Left(ex) }
```

Typeclass

Scala Future

Applicative
pure

Functor
map

Monad
flatMap

ApplicativeError
handleError

로직은 비슷해하지만 API가 조금씩 다르다

```
def getItem(id: Long): MonixTask[Either[Throwable, ItemDto]] =  
  if(id < 0) MonixTask.pure(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .onErrorRecover { case ex: Throwable => Left(ex) }
```

```
def getItem(id: Long): ScalaFuture[Either[Throwable, ItemDto]] =  
  if(id < 0) ScalaFuture.successful(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .recover { case ex: Throwable => Left(ex) }
```

Twitter Future

```
def getItem(id: Long): TwitterFuture[Either[Throwable, ItemDto]] =  
  if(id < 0) TwitterFuture.value(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .handle { case ex: Throwable => Left(ex) }
```

우리는 cats의 typeclass를 통해서

- scala.concurrent.Future
- com.twitter.util.Future
- monix.eval.Task
- cats.effect.IO

서로 다른 라이브러이지만 같은 API를 사용할수 있다.

기존의 코드, scala Future의 API를 쓰고 있다.

```
def getItem(id: Long): ScalaFuture[Either[Throwable, ItemDto]] =  
  if(id < 0) ScalaFuture.successful(  
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))  
  else  
    itemRepository.findById(id)  
      .flatMap(item =>  
        optionRepository.findById(item.id)  
          .map(option =>  
            Right(ItemDto(item.id, item.name, option)))  
      .recover { case ex: Throwable => Left(ex)}
```

cats API로 코드 구현하기

```
def getItem[F[_]](id: Long)
  (implicit F: MonadError[F, Throwable]): F[Either[Throwable, ItemDto]] =
  if(id < 0) F.pure(
    Left(new IllegalArgumentException(s"유효하지 않은 ID입니다. $id")))
  else
    itemRepository[F].findById(id)
      .flatMap(item =>
        optionRepository[F].findItemId(item.id)
          .map(option =>
            Either.right[Throwable, ItemDto](ItemDto(item.id, item.name, option)))
      .handleError(ex => Left(ex))
```

pure, flatMap, map, handleError
모두 cats의 api를 사용하였다.

사용하는 곳에서 원하는 Effect를 주입

이 함수는 MonadError에 대한 instance를 가지고 있는 다양한 monadic data type을 수용하는 유연한 구조가 되었다.

```
// before - only scala future
def getItem(id: Long): ScalaFuture[Either[Throwable, ItemDto]]
// after
def getItem[F[_]](id: Long)(implicit F: MonadError[F, Throwable])
: F[Either[Throwable, ItemDto]]
```

// getItem을 호출하는 시점에 타입을 지정해주면 된다.

// 그러면 ScalaFuture, TwitterFuture 그리고 MonixTask가 cats api를 통해서 실행된다.

```
val scalaFuture = getItem[ScalaFuture](10)
val twiiterFuture = getItem[TwitterFuture](10)
val monixTask = getItem[MonixTask](10)
```

서로의 라이브러리인데 같은 API

자바의 CompletableFuture
왜 이렇게 설계했는지 모르겠다...

```
// java completable future
val a = CompletableFuture.completedFuture(10)
// map
val b: CompletableFuture[Int] = a.thenApply(x => x + 10)
// flatMap
val c = b.thenCompose(x => CompletableFuture.completedFuture(x * 10))
```

난 암기 과목이 싫다.

thenApply, thenCompose 함수를 매번 검색해서 알아낸다.

The screenshot shows a Stack Overflow question page. The title is "What is CompletableFuture's equivalent of flatMap?". Below the title, there is an advertisement for Stack Overflow Jobs featuring chess pieces and the text "Make your next move with a career site that's by developers". The question itself asks if it's possible to convert a CompletableFuture<CompletableFuture<byte[]>> type to a CompletableFuture<byte[]>. The code provided uses thenApply and thenCompose methods to achieve this conversion. The code is as follows:

```
public Future<byte[]> convert(byte[] htmlBytes) {
    PhantomPdfMessage htmlMessage = new PhantomPdfMessage();
    htmlMessage.setId(UUID.randomUUID());
    htmlMessage.setTimestamp(new Date());
    htmlMessage.setEncodedContent(Base64.getEncoder().encodeToString(htmlBytes));

    CompletableFuture<CompletableFuture<byte[]>> thenApply = CompletableFuture.supply
        worker -> worker.convert(htmlMessage).thenApply(
            pdfMessage -> Base64.getDecoder().decode(pdfMessage.getEncodedContent())
        );
    }

}
```

Tags at the bottom of the post include java, java-8, and completable-future.

실습 - Monad instance 만들기

```
// CompletableFuture 모나드 instance 만들기.  
implicit val futureInstance = new Monad[CompletableFuture]  
  with StackSafeMonad[CompletableFuture] {  
  
  def pure[A](x: A): CompletableFuture[A] =  
    CompletableFuture.completedFuture(x)  
  
  def flatMap[A, B]  
    (fa: CompletableFuture[A])  
    (f: A => CompletableFuture[B]): CompletableFuture[B] =  
    fa.thenCompose(f.asJava)  
}
```

Native API vs Cats API

비교하여 보자.

```
// Java completable future
val a = CompletableFuture.completedFuture(10)
val b = a.thenApply(x => x + 10)
val c = b.thenCompose(x => CompletableFuture.completedFuture(x * 10))
```

```
// Cats monad api
val a2 = 10.pure[CompletableFuture]
val b2 = a2.map(x => x + 10)
val c2 = b2.flatMap(x => (x * 10).pure[CompletableFuture])
```

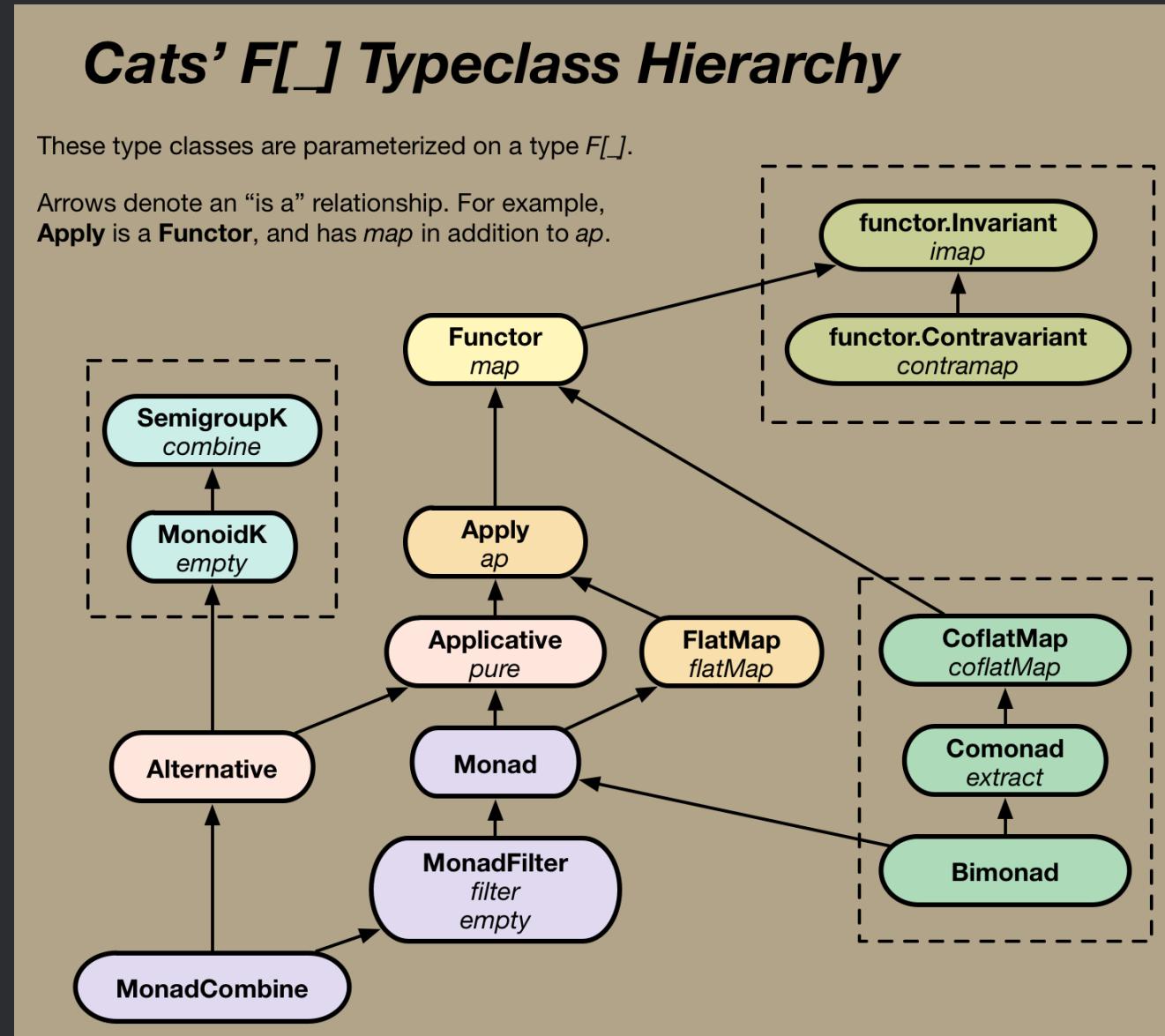
개념에 너무 억매이지 말자.

Future, HashMap, HttpServer 구현하진 못해도 잘 쓰고 있다.

우리는 더 어려운 API도 잘 쓰고 있었다.

Cats API가 가져다주는 간결함과 실용성에 대해서 알아보자.

조금 예전버전(2015) 이지만 이게 가볍게 보기에는 좋네...¹²



¹² <https://github.com/typelevel/cats/issues/95#issuecomment-114023955>

Cats의 구성 - Typeclass

그러나 많다고 쫄 필요없다. 😱

시작하는 단계라면 Functor와 Monad만 알아도 된다.

```
trait Functor[F[_]] {  
    def map[A, B](fa: F[A])(f: A => B): F[B]  
}
```

```
trait Monad[F[_]] {  
    def pure[A](a: A): F[A]  
    def flatMap[A, B](fa: F[A])(f: A => F[B]): F[B]  
}
```

Cats의 구성 - Data types

역시나 많다고 졸 필요없다. 😮

시작하는 단계라면 OptionT와 EitherT만 알아도 된다.

- OptionT = monad transformer for Option⁵
OptionT[F[_], A] is a light wrapper on an F[Option[A]]
- EitherT = monad transformer for Either⁶
EitherT[F[_], A, B] is a lightweight wrapper for F[Either[A, B]]

⁵ <https://typelevel.org/cats/datatypes/optiont.html>

⁶ <https://typelevel.org/cats/datatypes/either.html>

Cats의 구성 - Instance

scala stdlib에서 제공하는 data type에 대한 cats typeclass의 instance를 제공

```
// .../cats/instances/all.scala
trait AllInstances
  extends AnyValInstances
  with ...
  with ListInstances
  with MapInstances
  with OptionInstances
  with OrderInstances
  with OrderingInstances
  with TupleInstances
  with UUIDInstances
  with VectorInstances
  with ...
```

그리고 cats의 data type에 대한 instance도 제공해줌

Do It Yourself 정신!

구글링을 하다보니 monad transformer라는 개념이 나오고

cats에 있는지 모르고 😊

2016년 9월 25일 블로그에 있는 코드 줍기를 하며 monad transformer 직접 만들어 사용한다...²⁰

²⁰ <http://loicdescotte.github.io/posts/scala-compose-option-future/>

Future Option의 Monad Transformer 추가

master v0.3.9 ... 0.0.6

liam-m committed on Apr 25, 2016 patch diff

Showing 13 changed files with 331 additions and 13 deletions.

```
import scala.concurrent.{ExecutionContext, Future}
case class FutureOption[+A](future: Future[Option[A]]) extends AnyVal {
  def flatMap[B](f: A => FutureOption[B])(implicit ec: ExecutionContext): FutureOption[B] = {
    val newFuture = future.flatMap {
      case Some(a) => f(a).future
      case None     => Future.successful(None)
    }
    FutureOption(newFuture)
  }

  def map[B](f: A => B)(implicit ec: ExecutionContext): FutureOption[B] = {
    FutureOption(future.map(option => option.map(f)))
  }

  def filter(p: (A) => Boolean)(implicit ec: ExecutionContext): FutureOption[A] = filterWith(p)

  def filterWith(p: (A) => Boolean)(implicit ec: ExecutionContext): FutureOption[A] = {
    FutureOption(future.map {
      case Some(a) => if (p(a)) Some(a) else None
      case _        => None
    })
  }
}
```

Monad Transformer 공식

- 함수의 리턴타입을 정하고 구현을 시작하라.
- 내부에서 호출하는 함수의 결과를 적절하게 lift하라.
- Future[Option[A]]를 반환한다면 무조건 OptionT를 고려해야 한다.

3. 의문 - Monad Transformer가 왜 필요한가?

Monad는 compose를 통해서 새로운 Monad를 만들수 없다. 🤔
Applicative에서만 정의가 가능한 함수이기 때문이다.²²

```
trait Applicative[F[_]] {
  def compose[G[_]: Applicative]: Applicative[λ[α => F[G[α]]]] =
    new ComposedApplicative[F, G] {
      val F = self
      val G = Applicative[G]
    }
}
Monad[Future].compose[Option] // Applicative[Future[Option[α]]]
```

²² <https://github.com/typelevel/cats/blob/2ae785f726b810438fa5b429d5c9d28f1be2e69d/core/src/main/scala/cats/Applicative.scala#L88-L92>

3. 의문 - Monad Transformer가 왜 필요한가?

모나드가 합성을 할수 없기 때문에
Future[Option[A]]을 나타내는
Monad를 바로 만들수 없지만
Monad Transformer를 통해서
합성한것과 같은 효과를 얻을수 있다.²¹

²¹ <http://book.realworldhaskell.org/read/monad-transformers.html>

Cats **Monad**가 가지고 있는 다양한 함수들

일반적인 함수가 50여개 $\text{XXX}2\sim22$ 모양을 가진 함수가 60개 합쳐서 110여개쯤 된다.

```
// scala reflection으로 함수 뽑아보기
import scala.reflect.runtime.universe._

typeOf[Monad[Future]].members.map(show(_))

// 결과
widen, whileM_, whileM, whenA, void, untilM_,
untilM, unlessA, unit, tupleRight, tupleLeft,
replicateA, product, productREval, productR,
productLEval, productL, point, mproduct, map, lift,
iterateWhileM, iterateWhile, iterateUntilM,
iterateUntil, imap, ifM, fproduct, forEffect,
forEffectEval, followedBy, followedByEval, fmap,
flatten, flatTap, compose, composeFunctor,
composeContravariant, composeContravariantMonoidal,
composeApply, as, ap, <*, <*>, *>,
tuple2~22, map2Eval, map2~22, ap2~22
```

cache miss에 사용한 코드들

```
import cats.effect.Effect
// local cache에서
def fetchLocalCache[F[_]](key: Int)(implicit F: Effect[F]): F[Option[User]]
// remote cache에서
def fetchRemoteCache[F[_]](key: Int)(implicit F: Effect[F]): F[Option[User]]
// persistence db에서
def fetchDB[F[_]](key: Int)(implicit F: Effect[F]): F[Option[User]]
```

