Improving Your Code With Union Types

What is "good code"?


YEAH, WELL, YOU KNOW,
THAT'S JUST, LIKE.. UH
YOUR OPINION, MAN
quickmeme.com

Commonly accepted attributes

- Expressive
- Easier to change
- Easier to read
- More modular
- More correct

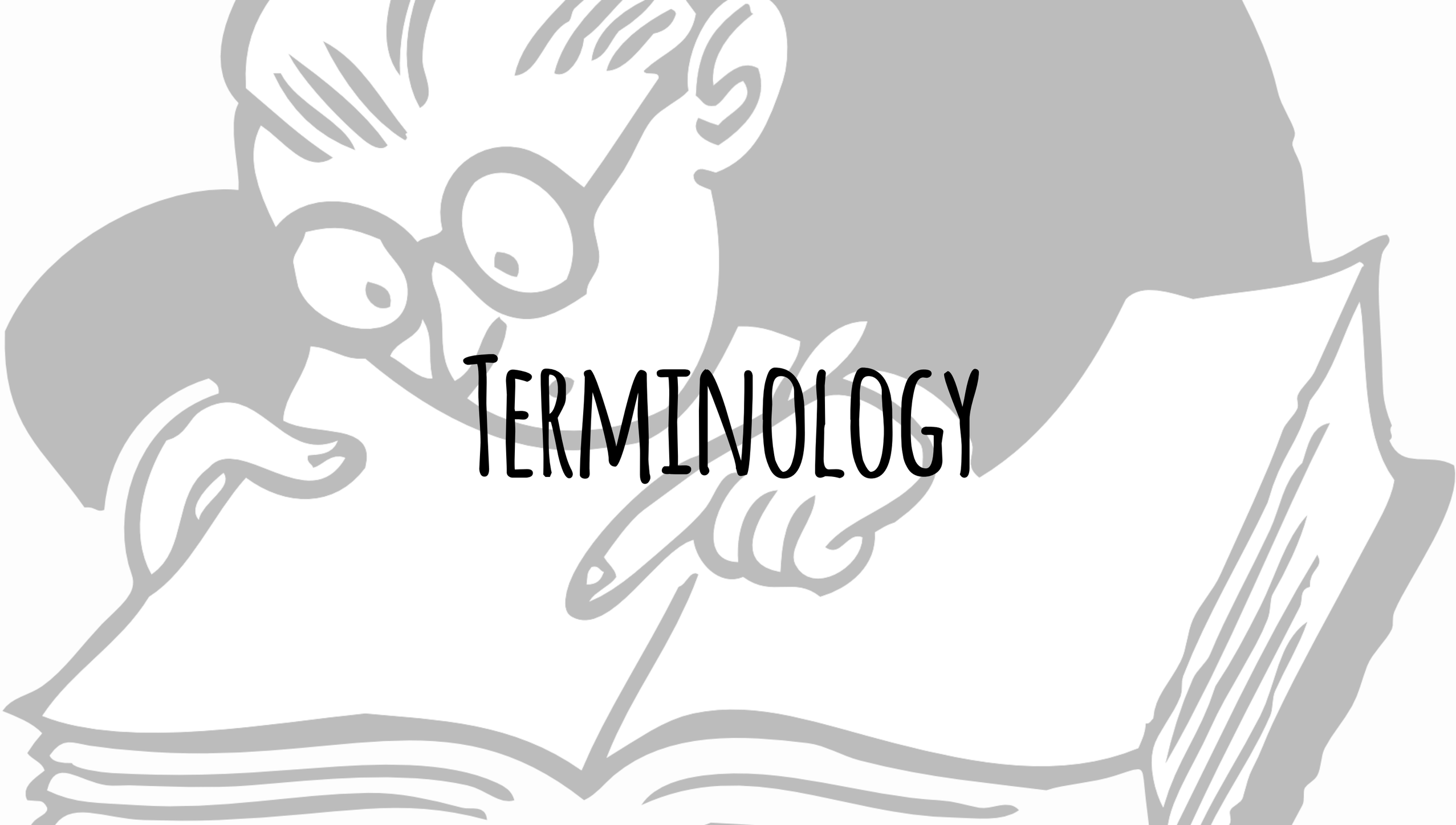
JavaScript	Elm
`3`	`3`
`3.1415`	`3.1415`
`"Hello world!"`	`"Hello world!"`
Multiline strings not widely supported	`"""multiline string"""`
`'Hello world!'`	Cannot use single quotes for strings
No distinction between characters and strings	`'a'`
`true`	`True`
`[1,2,3]`	`[1,2,3]`

Objects / Records

JavaScript	Elm
`{ x: 3, y: 4 }`	`{ x = 3, y = 4 }`
`point.x`	`point.x`
`point.x = 42`	`{ point | x = 42 }`

Functions

JavaScript	Elm
`function(x, y) { return x + y; }`	`\x y -> x + y`
`Math.max(3, 4)`	`max 3 4`
`Math.min(1, Math.pow(2, 4))`	`min 1 (2^4)`



Terminology

Sum types (AKA Enums)

```
type Suit
  = Hearts
  | Spades
  | Diamonds
  | Clubs
```

4 DIFFERENT SUITS

Product types (AKA Tagged values)

```
type Card = Card Value Suit
```

13 values x 4 Suits = 52

JOKER

JOKER

Combining them together

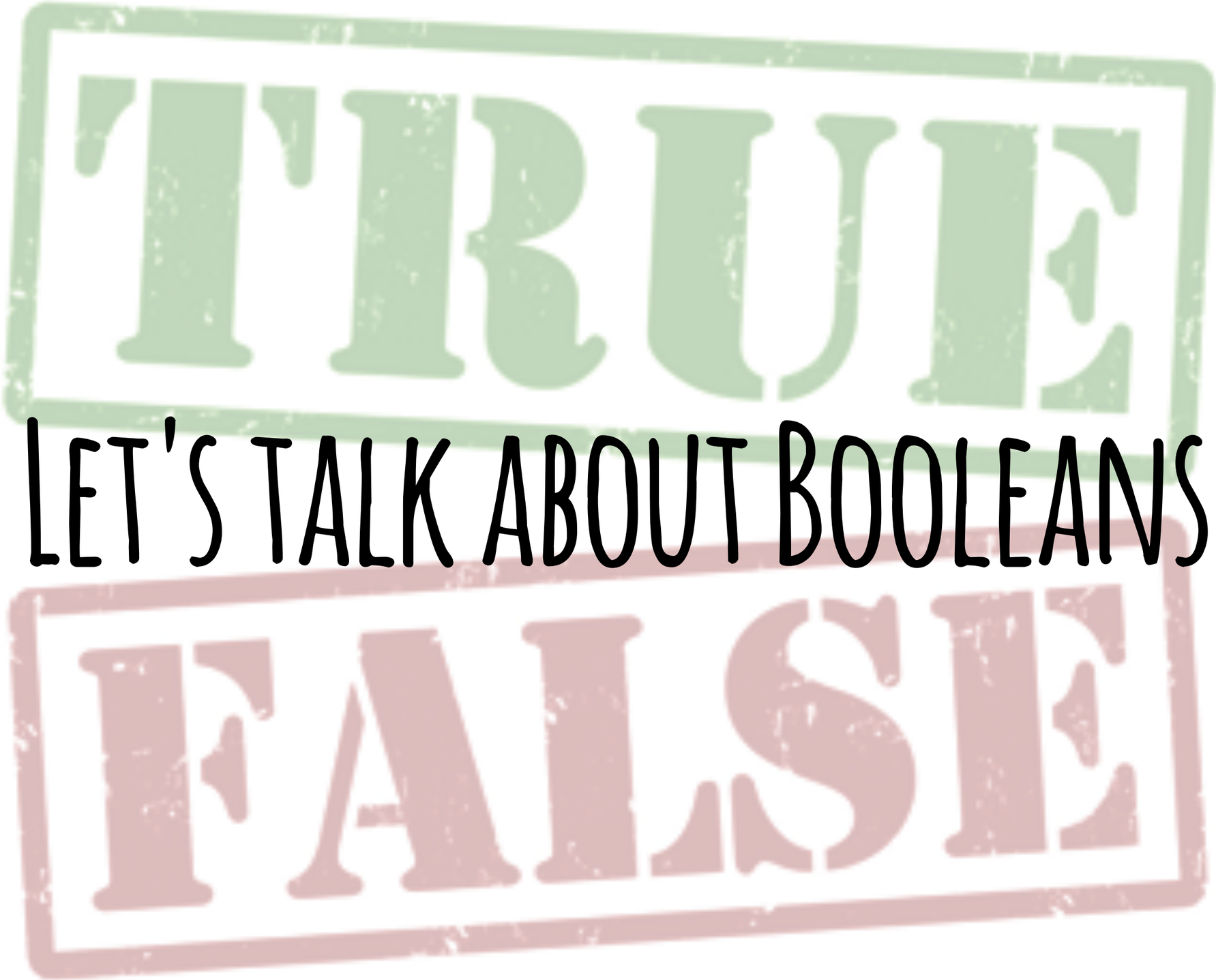
```
type Coloring = Colored | Grayscale

type Card
  = Card Value Suit
  | Joker Coloring
```

$$(13 \times 4) + 2 = 54$$

Type variables

```
type Maybe a
  = Nothing
  | Just a
```

What are they good for?



Let's talk about Booleans

Three-State Boolean problem

```
type alias Order =
  { id : Int
  , delivery : Maybe Bool
  }
```

Three states

1. Just True
2. Just False
3. Nothing

Sum type

```
type Delivery = Delivery | Pickup | NoneSelected

type alias Order =
  { id : Int
  , delivery : Delivery
  }
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ❌ More correct


TERENCE HILL
BUD SPENCER
DOUBLE TROUBLE

Double Dependant Boolean

```
type alias Order =
  { id : Int
  , delivery : Bool
  , thirdParty : Bool
  }
```

Four states

- True True
- True False
- False True
- False False

What does this even mean

```
{ id = 1
, delivery = False
, thirdParty = True
}
```

Sum type

```
type Delivery
  = Delivery
  | Pickup
  | ThirdPartyDelivery
  | NotSelected

type alias Order =
  { id : Int
  , delivery : Delivery
  }
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ❌ More correct

Boolean Flags

Less readable

```
npc : Character
npc =
  Character True True False
```

More readable

```
npc : Character
npc =
  Character Immortal Female Stationary
```

Looks like

```
type Mortality = Mortal | Immortal
type Gender = Male | Female
type Mobility = Stationary | Mobile

type alias Character =
  { mortality : Mortality
  , gender : Gender
  , mobility : Mobility
  }
```

More extensible

```
type Gender = Female | Male | NonBinary
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ❌ More correct


Why not Strings?

How Many Mobilities?

```
type Mobility = Mobile | Stationary
```

How Many Mobilities?

```
type alias Mobility = String
```

Case statements

```
move : Int -> Character -> Character
move distance character =
  case character.mobility of
    "Mobile" ->
      { character | position = character.position + distance }

    "Stationary" ->
      character
    _ ->
      character
```

Sum type gives error

```
-- MISSING PATTERNS ----------------------------------------------

This `case` does not have branches for all possibilities.

11|>    case character.mobility of
12|>      Mobile ->
13|>        { character | position = character.position + distance }
14|>
15|>      Stationary ->
16|>        character

You need to account for the following values:

    Flying
```

Typos

```
npc : Character
npc =
  Character "Mobil" 55
```

Sum type gives error

```
-- NAMING ERROR ---------------------------------------------------------

Cannot find variable `Mobil`

20|   Character Mobil 55
                ^^^^^
Maybe you want one of the following?

    Mobile
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ✅ More correct

Primitive Obsession

Dangerous primitives

```
type alias User =
  { name : String
  , age : Int
  , bankBalance : Int
  , salary : Int
  }
```

Working with Cash

```elm
addBalance : User -> Int -> User
addBalance user amount =
  { user | bankBalance = bankBalance + amount }
```

Invalid operation

```
payday : User -> User
payday user =
  addBalance user user.age
```

Custom type

```
type Dollar = Dollar Int

type alias User =
  { name : String
  , age : Int
  , bankBalance : Dollar
  , salary : Dollar
  }
```

Using Dollar

```
addBalance : User -> Dollar -> User
addBalance user dollarAmount =
  let
      (Dollar balance) = user.bankBalance
      (Dollar amount) = dollarAmount
  in
      { user | bankBalance = Dollar (balance + amount) }
```

Invalid operation

```
payday : User -> User
payday user =
  addBalance user user.age
```

Compiler error

```
-- TYPE MISMATCH ---------------------------------------------------------

The 2nd argument to function `addBalance` is causing a mismatch.

21|     addBalance user user.age
                     ^^^^^^^^

Function `addBalance` is expecting the 2nd argument to be:

    Dollar

But it is:

    Int
```

Pain points

1. Unwrap value
2. Add the integers
3. Re-wrap value

Sound familiar?

Mapping

1. Unwrap value
2. Apply function
3. Re-wrap value

Mapping Maybe

```
doubleMaybe : Maybe Int -> Maybe Int
doubleMaybe maybe =
  Maybe.map (\n -> n * 2 ) maybe
```

Summing two Maybes

```elm
sumMaybe : Maybe Int -> Maybe Int -> Maybe Int
sumMaybe m1 m2 =
  Maybe.map2 (+) m1 m2
```

We need that

If only there was a `Dollar.map2`

Mapping to the rescue

```elm
module Dollar exposing (Dollar, map2)

map2 : (Int -> Int) -> Dollar -> Dollar -> Dollar
map2 function (Dollar d1) (Dollar d2) =
  Dollar (function d1 d2)
```

Using Dollar

```
addBalance : User -> Dollar -> Dollar
addBalance user amount =
  { user | bankBalance = Dollar.map2 (+) user.bankBalance amount }
```

Getting clever

```
module Dollar exposing (Dollar, map2)

sum : Dollar -> Dollar -> Dollar
sum =
  map2 (+)
```

General utility

- map
- map2
- sum
- subtract
- multiply
- divide

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ✅ More correct

Opaque types

Start with this

```
module Point exposing (Point)

type alias Point = { x : Int, y : Int }
```

Use in in Code

```
path : List Point
path =
  [ Point 1 1,
  , Point 100 100
  ]
```

Now you want to add 3D

```
module Point exposing (Point)

type alias Point = { x : Int, y : Int, z : Int}
```

Broken code

Wrapping in a type

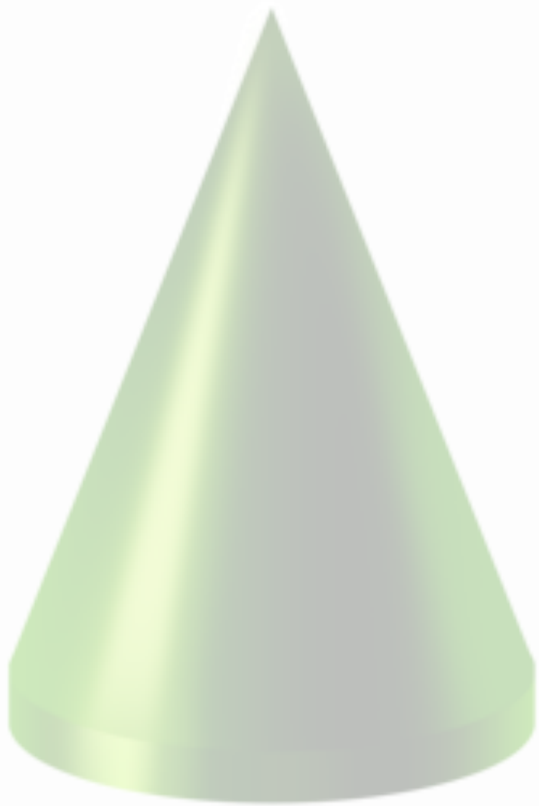
```
module Point exposing (Point, fromXY)

type Point = Point { x : Int, y : Int }

fromXY : (Int, Int) -> Point
fromXY (x, y) =
  Point { x = x, y = y }
```

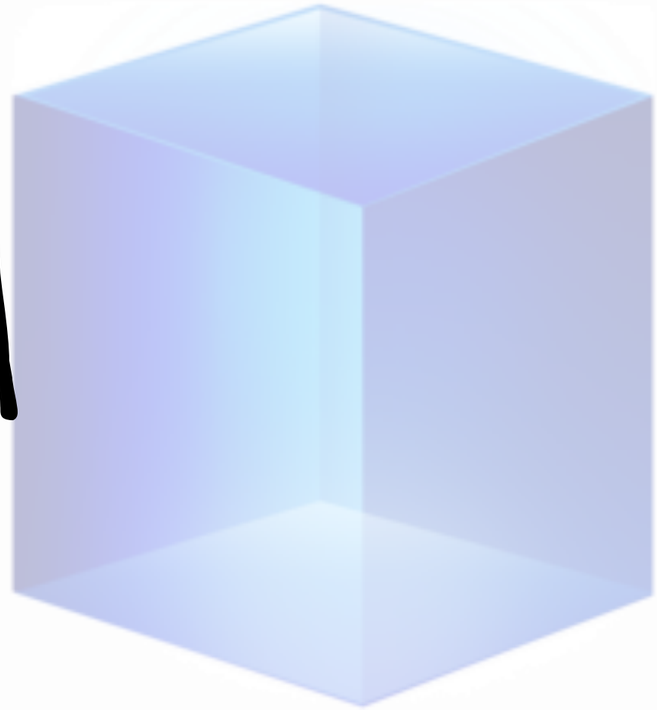
Adding 3D

```
module Point exposing (Point, fromXY)

type Point = Point { x : Int, y : Int, z : Int }

fromXY : (Int, Int) -> Point
fromXY (x, y) =
  Point { x = x, y = y, z = 0 }
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ✅ More modular
- ❌ More correct



The shape of data

Primitives

```
type alias Card =
  { suit : String
  , value : Int
  }
```

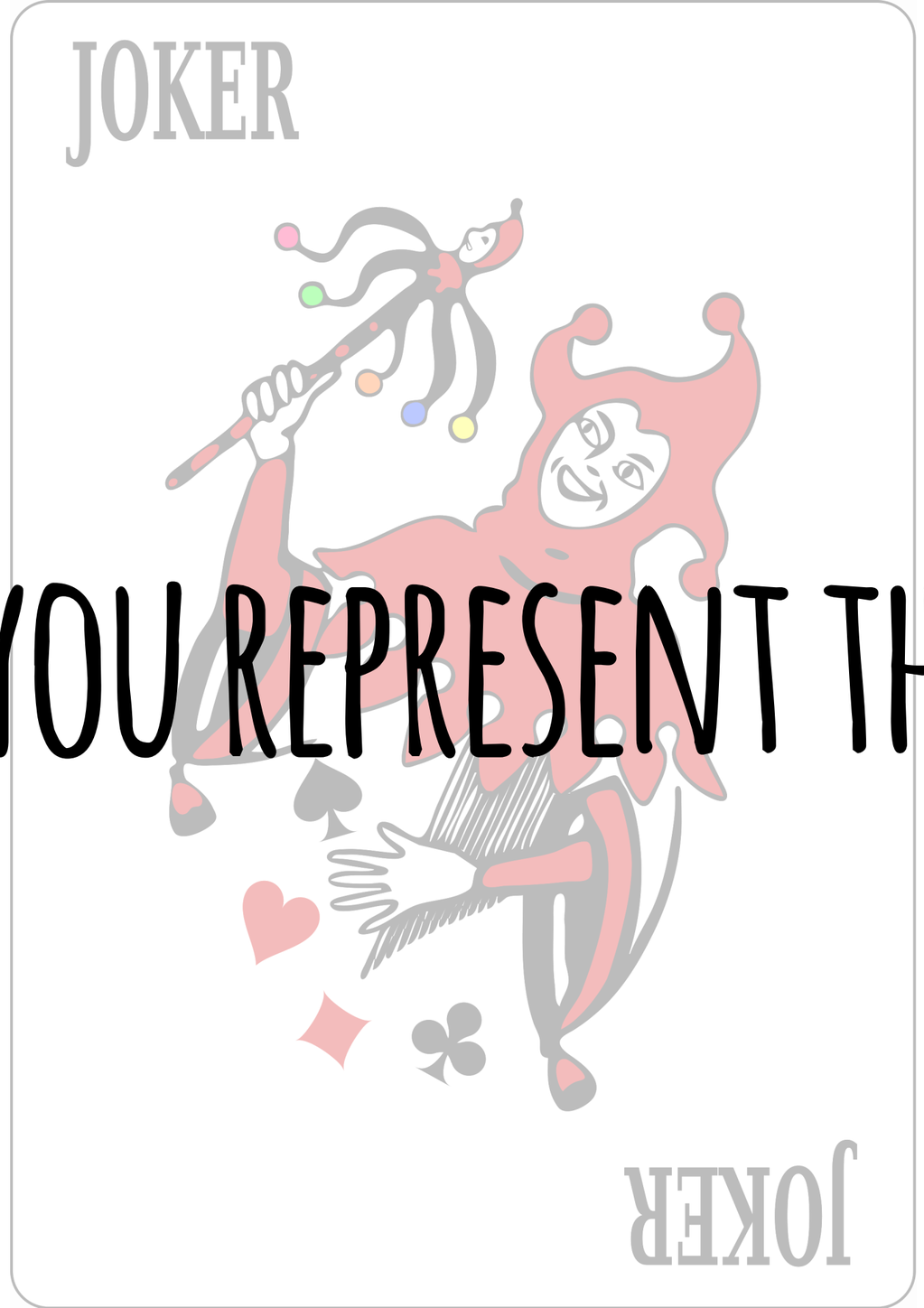
Looks like

```
Card "Hearts" 2
```

How Many of this type of card?

This is valid

```
Card "Starfish" 1999
```



How do you represent the Joker?

You have to force it

```
Card "Joker" 0
```

or

```
Card "Joker" Nothing
```

Union types allow multiple shapes

```
type Card
   = Card Value Suit
   | Joker Coloring
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ✅ More correct
- ✅ More expressive

HTTP

Maybe

```
type alias Model =
  { lastOrder : Maybe Order
  }
```

Explore the shape of data

```
type RemoteData e a
    = NotAsked
    | Loading
    | Failure e
    | Success a
```

Improved Model

```
type alias Model =
  { lastOrder : RemoteData Http.Error Order
  }
```

Improvements

- ✅ Easier to change
- ✅ Easier to read
- ❌ More modular
- ❌ More correct
- ✅ More expressive

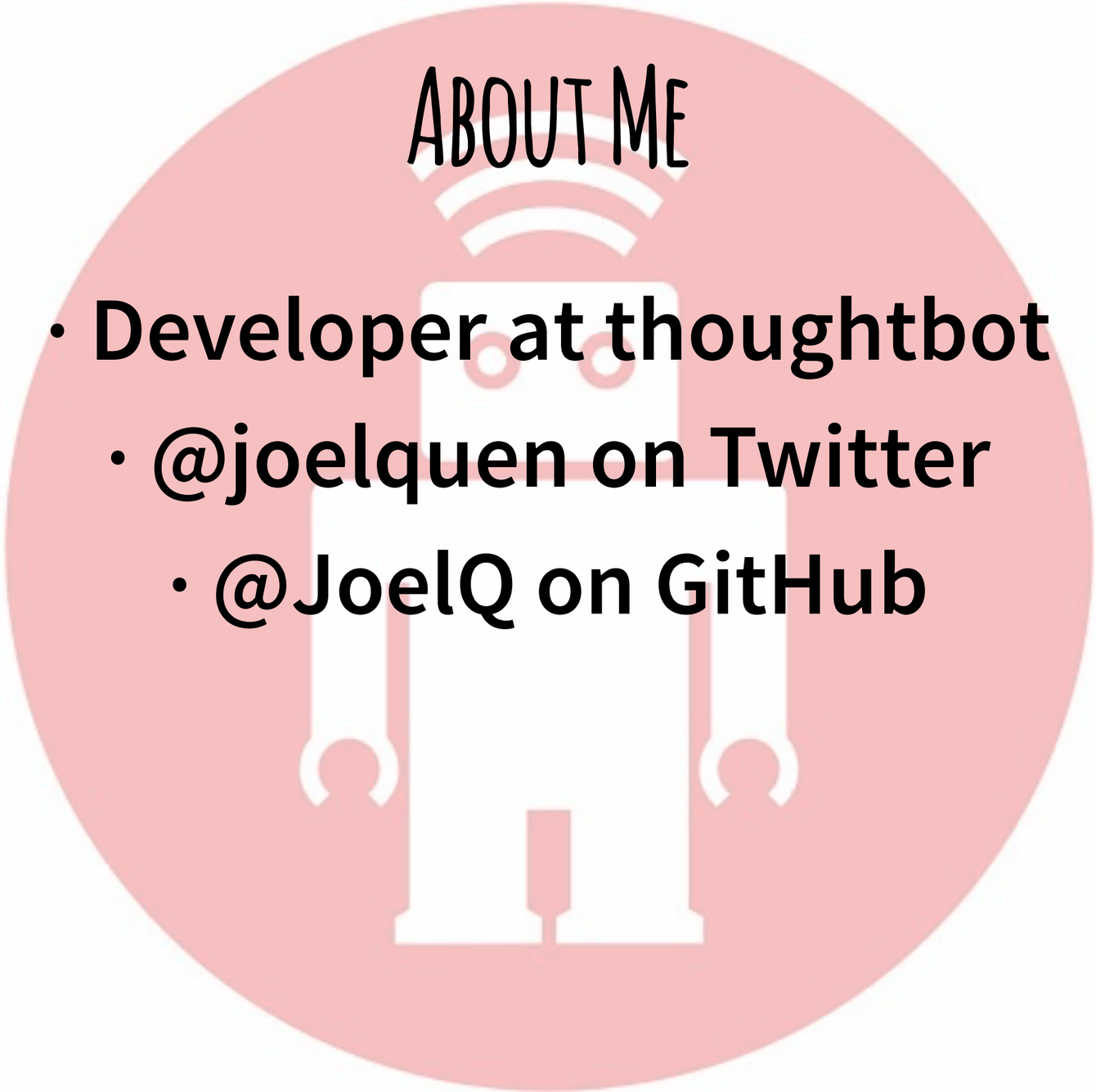
Improve your code with Union Types

Questions?

Slides are on GitHub

https://github.com/JoelQ/adts-elm-meetup

About Me



- Developer at thoughtbot
- @joelquen on Twitter
- @JoelQ on GitHub