A Type Driven Approach to Functional Design

Michael Feathers
Groupon

A Type Driven Approach to Functional Design

(just because)

Michael Feathers
Groupon

```ruby
# :: [event] -> [month,int]
def avg_lines_per_commit_by_month events
  cls_by_month = lines_added_per_commit(events).group_by {|date,_| month_from_date(date) }
  cls_by_month.map {|_,cls| cls.map {|cl| cl[1]}.mean }.flatten
end

# :: [event] -> Float
def percent_reduction method_events
  non_deleted = method_events.select {|e| e.status != :deleted }
  return 0.0 if non_deleted.count == 0
  num_reductions = non_deleted.each_cons(2) \
                              .map {|before, after| after.method_length < before.method_length
                              .count(true)
  num_reductions / non_deleted.count.to_f
end

# :: [event] -> [FixNum]
def refactoring_reduction_profile events
  events.group_by(&:method_name) \
        .map {|_,e| percent_reduction(e) } \
        .freq_by {|e| (e * 100 / 10).to_i  }
end
```

a

[a]

```
map :: (a -> b) -> [a] -> [b]
```

```
map :: (a -> b) -> [a] -> [b]

map (+2) [1,2,3]
```

region 7 9 “expertsexchange”

```
region 7 9 "expertsexchange"

region :: Int -> Int -> String -> String
```

regionFrom7 9 “expertsexchange”

regionFrom7 :: ~~Int ->~~ Int -> String -> String

Hoping

Line Break algorithm

String -> String

String -> [String] -> [[String]] -> [String] -> String

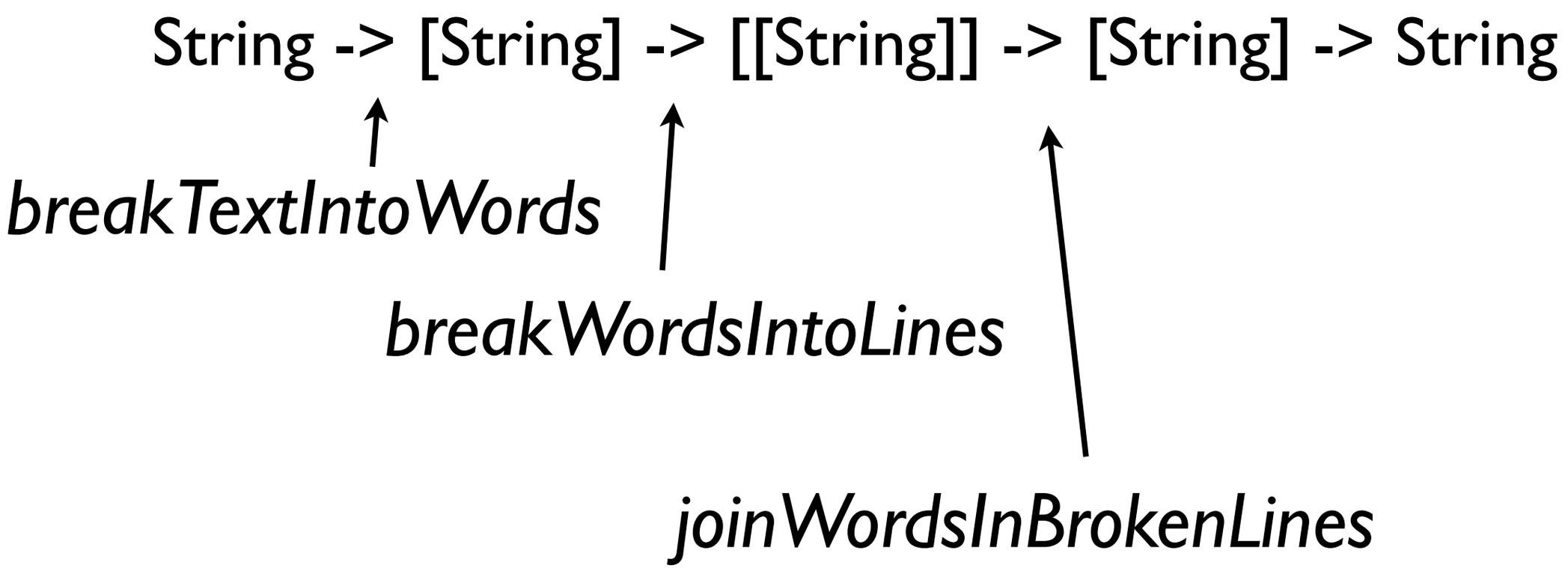
```
String -> [String] -> [[String]] -> [String] -> String
```

breakTextIntoWords

String -> [String] -> [[String]] -> [String] -> String

breakTextIntoWords

breakWordsIntoLines


String -> [String] -> [[String]] -> [String] -> String
breakTextIntoWords
breakWordsIntoLines
joinWordsInBrokenLines

String -> [String] -> [[String]] -> [String] -> String

breakTextIntoWords

breakWordsIntoLines

joinWordsInBrokenLines

joinBrokenLines

String -> [String] -> [[String]] -> [String] -> String

String -> [String] -> [[String]] -> [String] -> String

String -> [String] -> (Int -> [[String]]) -> [[String]] ->
[String] -> String

String -> [String] -> [[String]] -> [String] -> String

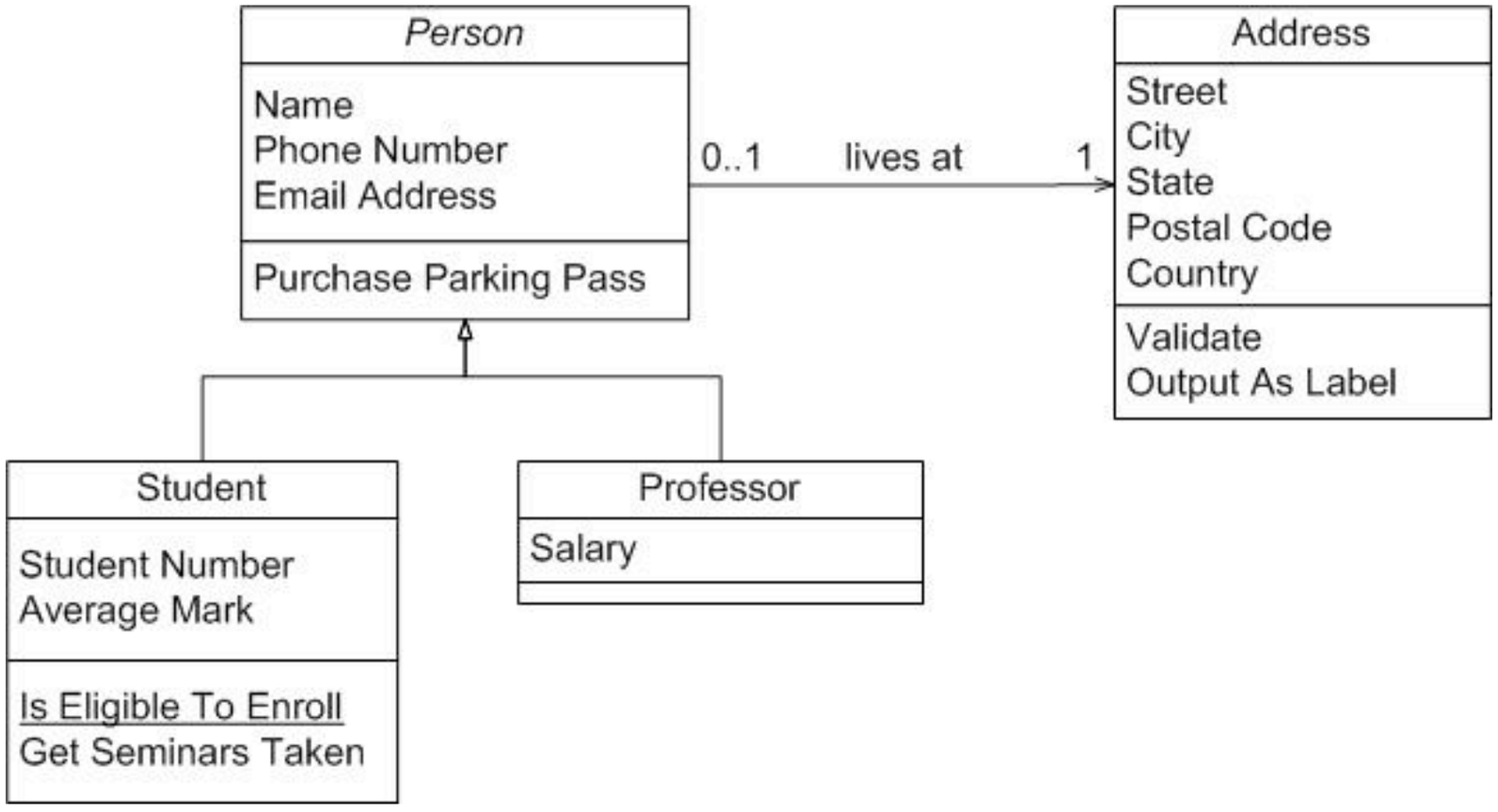
String -> [String] -> (Int -> [[String]]) -> [[String]] ->
[String] -> String

String -> [String] -> [[String]] -> [String] -> String

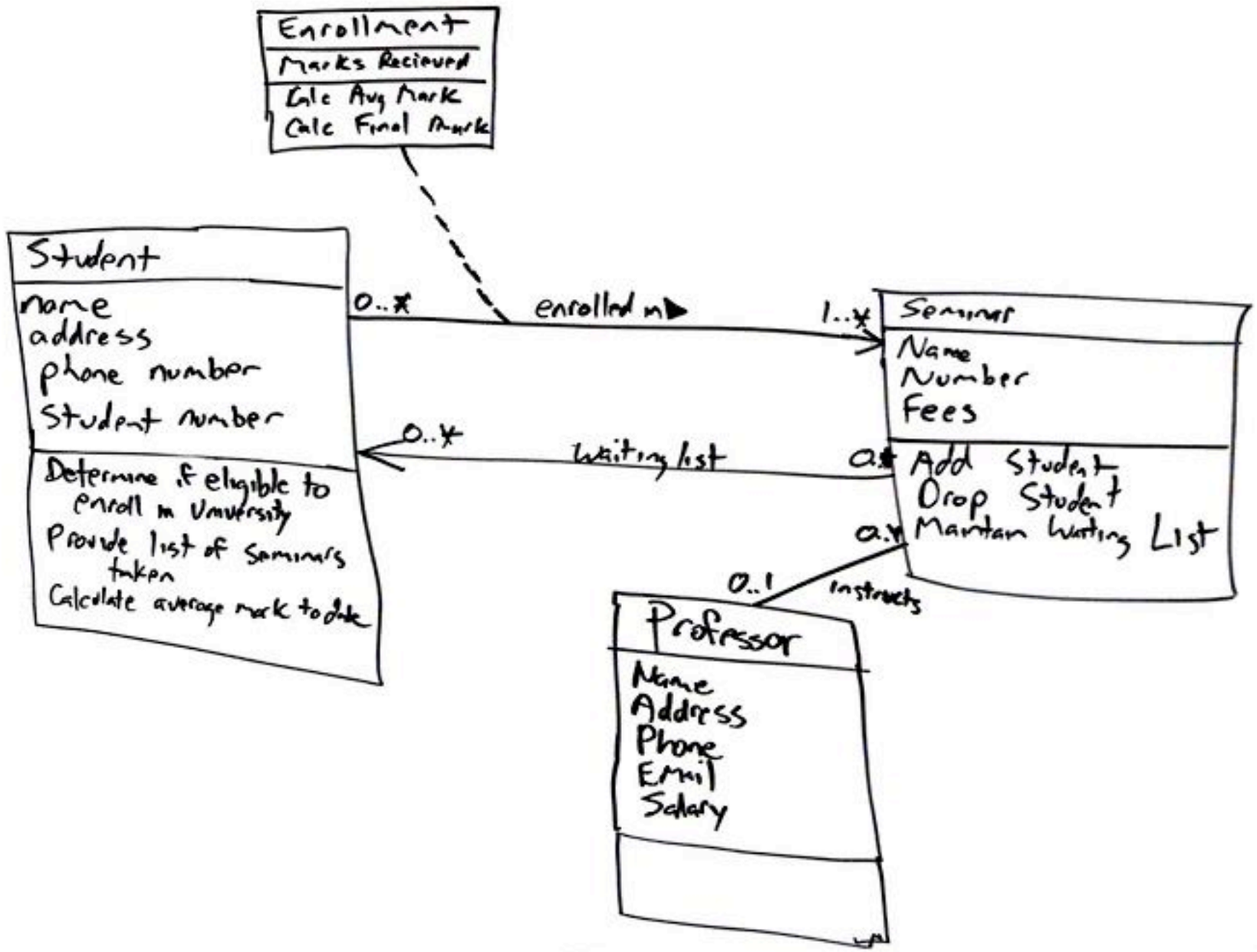
String -> [String] -> (Int -> [[String]]) -> [[String]] ->
[String] -> String

Abuse!

A Design Device



(via Scott Ambler)



(via Scott Ambler)

	A	B	C
1			
2		String	
3		[String]	
4		[[String]]	
5		[String]	
6		String	
7			
8			
9			

	A	B	C
1			
2		String	
3		[String]	
4		[[String]]	Int
5		[String]	
6		String	
7			
8			
9			

	A	B	C
1			
2		String	
3		[String]	
4		[[String]]	Int
5		[String]	
6		String	
7			
8			
9			
10		Int	
11			
12			

	A	B	C
1			
2		String	
3		[String]	
4		[[String]]	Int
5		[String]	
6		String	
7			
8			
9		Int -> [String]	
10		Int	
11			
12			
13			

	A	B	C
1			
2		String	
3	**breakTextIntoWords**	[String]	
4	**BreakWordsIntoLines**	[[String]]	Int
5	**joinWordsInBrokenLines**	[String]	
6	**joinLines**	String	
7			
8			
9		Int -> [String]	
10		Int	
11			

	A	B	C
1			
2		String	
3	**breakTextIntoWords**	[String]	
4	**BreakWordsIntoLines**	[[String]]	Int
5	**joinWordsInBrokenLines**	[String]	
6	**joinLines**	String	
7			
8			
9		Int -> [String]	
10		Int	
11			
12			

	A	B	C
1			
2		String	
3	**breakTextIntoWords**	[String]	
4	**BreakWordsIntoLines**	[[String]]	Int
5	**joinWordsInBrokenLines**	[String]	
6	**joinLines**	String	
7			
8			
9		Int -> [String]	
10	**brokenLinesWordCount**	Int	
11			
12			
13			

```
String -> [String] -> [[String]] -> [String] -> String
```

words

String -> [String] -> [[String]] -> [String] -> String

words

String -> [String] -> [[String]] -> [String] -> String

words

brokenLines

String -> [String] -> [[String]] -> [String] -> String

words

brokenLines

wordJoinedLines

String -> [String] -> [[String]] -> [String] -> String

words

brokenLines

wordJoinedLines

joinedLines

	A	B	C
1			
2		String	
3	**words**	[String]	
4	**brokenLines**	[[String]]	Int
5	**wordJoinedLines**	[String]	
6	**joinedLines**	String	
7			
8			
9		Int -> [String]	
10	**brokenLinesWordCount**	Int	
11			
12			

Report abuse

Theme: Vibrant Ink

```
1   import Data.List
2
3   lineBreak :: String -> String
4   lineBreak = joinedLines . wordJoinedLines . brokenLines . words
5
6   brokenLines :: [String] -> [[String]]
7   brokenLines [] = []
8   brokenLines wordList = brokenLine : brokenLines remainingWords
9     where (brokenLine, remainingWords) = splitAt (brokenLineWordCount wordList) wordList
10
11  brokenLineWordCount :: [String] -> Int
12  brokenLineWordCount = length . takeWhile (< 80) . scanl1 (+) . map wordLength
13
14  wordLength :: String -> Int
15  wordLength = succ . length
16
17  wordJoinedLines :: [[String]] -> [String]
18  wordJoinedLines = map (intercalate " ")
19
20  joinedLines :: [String] -> String
21  joinedLines = intercalate "\n"
```

```haskell
brokenLineWordCount :: [String] -> Int
brokenLineWordCount = length . takeWhile (< 80) . scanl1 (+) . map wordLength
```

```haskell
brokenLineWordCount :: [String] -> Int
brokenLineWordCount = length . takeWhile (< 80) . scanl1 (+) . map wordLength
```

[String] -> Int

```haskell
brokenLineWordCount :: [String] -> Int
brokenLineWordCount = length . takeWhile (< 80) . scanl1 (+) . map wordLength
```

[String] -> Int

[String] -> [Int] -> [Int] -> [Int] -> Int

(repeated types in endomorphic chain)

Affordances

Separate bins for your head and your notation

String -> [String] -> (Int -> [[String]]) -> [[String]] -> [String] -> String

Concentrates on Data

Concentrates on Data

..in naming

Favors combinator style

Staying in the same shape is the easiest way to get from here to there

Thank you