Functional Kats Conf 2015

Garth Gilmour
garth.gilmour@instil.co
@GarthGilmour

About Me (garth.gilmour@instil.co)

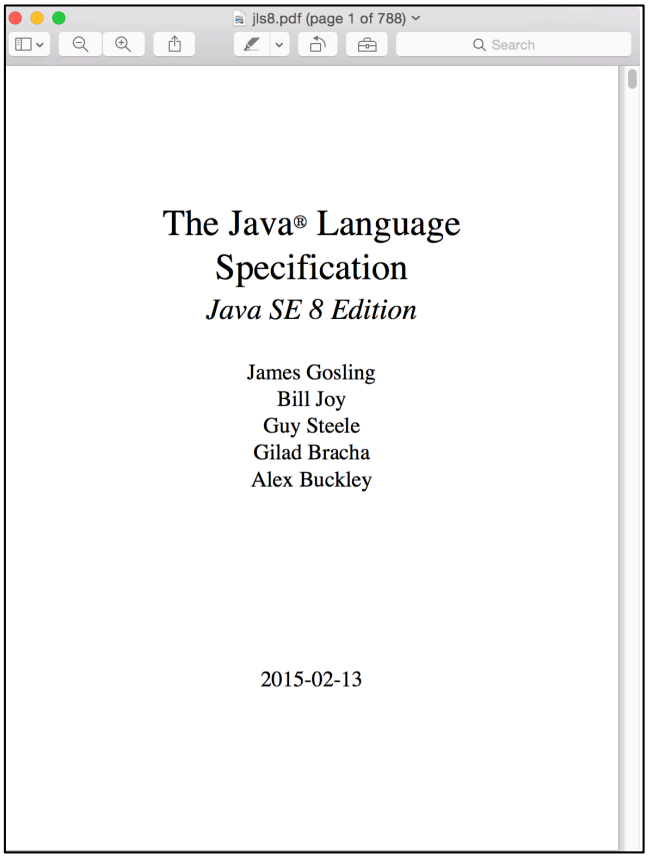
- **Experienced** trainer
  - 15 years in the trenches
  - Over 1000 deliveries
- The day job
  - Head of Learning at Instil
  - Enjoy the cheese…
- The night job(s)
  - Husband and father
  - Krav Maga Instructor
- Big Scala fanboy
  - Can we still be friends?

I Like Scala…

I Believe in Context…

What This Talk Is **Not** About (1)

What This Talk Is **Not** About (2)

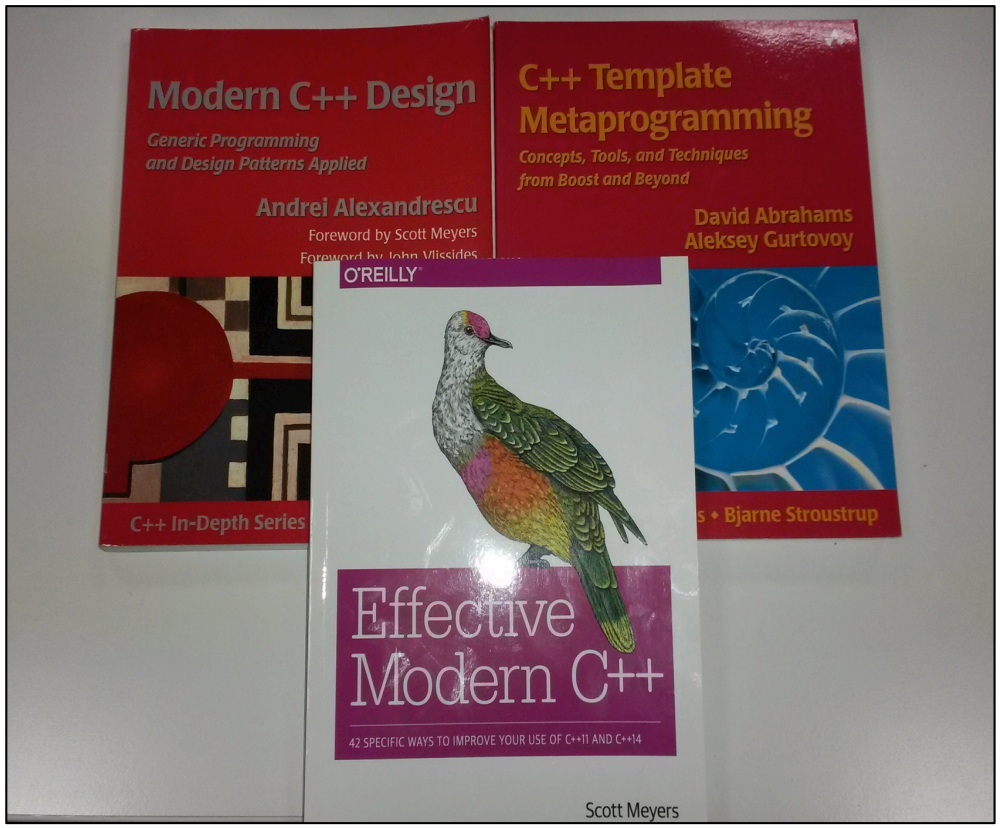


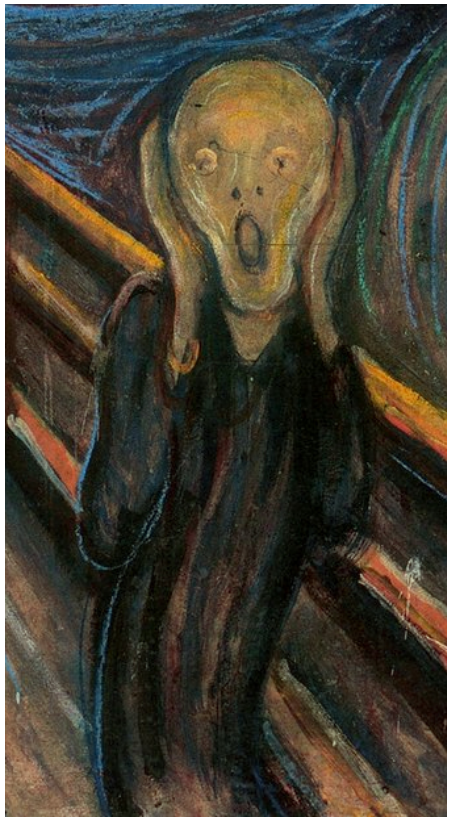

If you find yourself
in a fair fight,
YOUR
TACTICS
SUCK.
www.tacticaltshirts.com

However…


Sun Certification
Sun microsystems
We make the net work.
Garth Gilmour
HAS FULFILLED ALL REQUIREMENTS AS A
SUN CERTIFIED BUSINESS COMPONENT DEVELOPER
FOR THE JAVA™ 2 PLATFORM 1.3, ENTERPRISE EDITION
On July 7, 2003
Sun Certified
Scott McNealy, President, Chairman, and CEO of Sun Microsystems, Inc.
Terry Erdle, Vice President, Sun Educational Services
THE NETWORK IS THE COMPUTER ©Sun Microsystems, Inc., Sun, Sun Microsystems, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

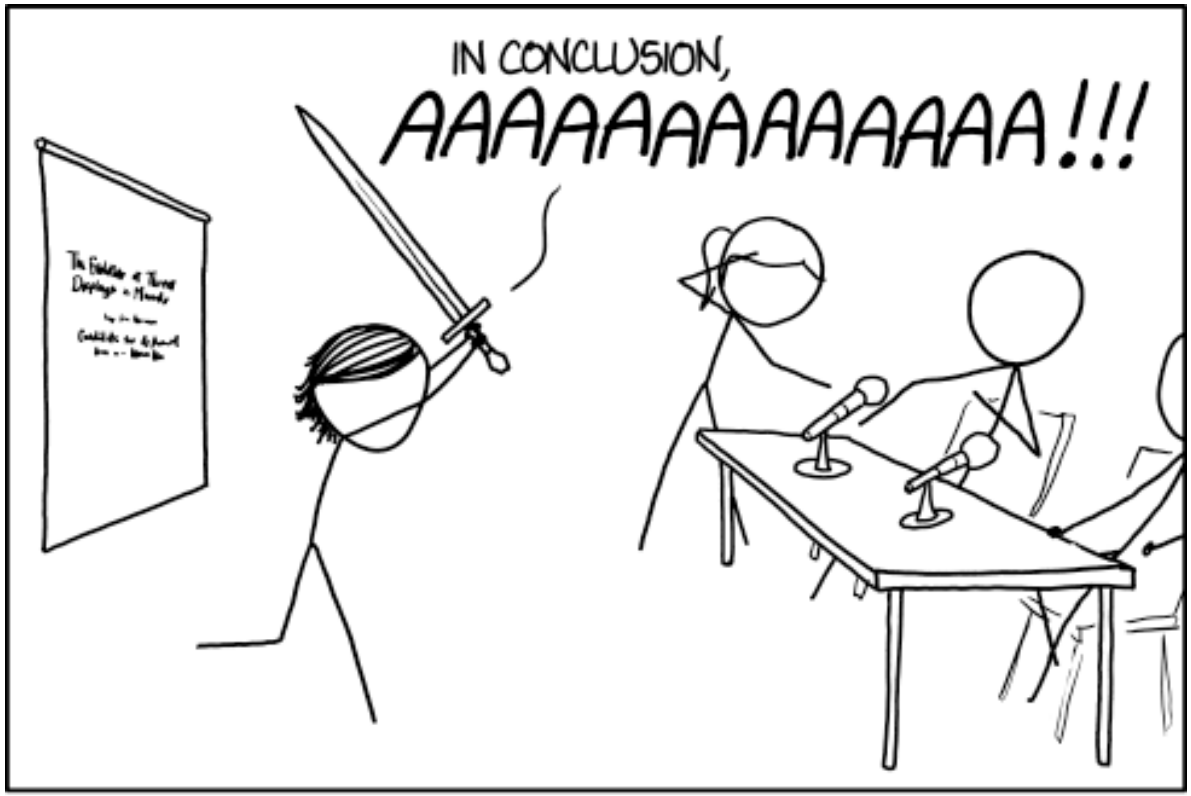
A Warning From History: C++






OUR FEATURE
PRESENTATION

My Thesis



THE BEST THESIS DEFENSE IS A GOOD THESIS OFFENSE.

Java Up To Version 4

Java in Version 8

Java 8 is More Complex Than ‘X’

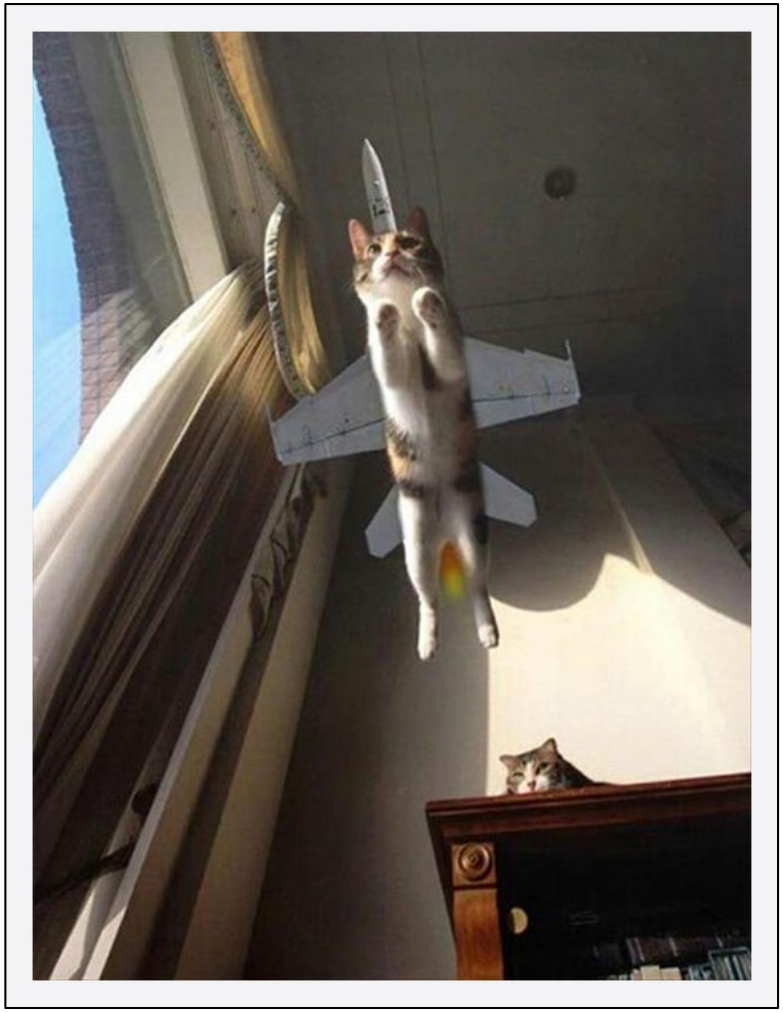
- Is NOT about absolute complexity
- Its about accidental vs. essential complexity
- Other languages have higher essential complexity
  - But the features ‘hang together’ and make sense

Complexity in Java 8

Primitive types and boxing
and arrays and collections
and non-reified generics
and var-args and lambdas
and method refs. Oh My!

Java In Version 8

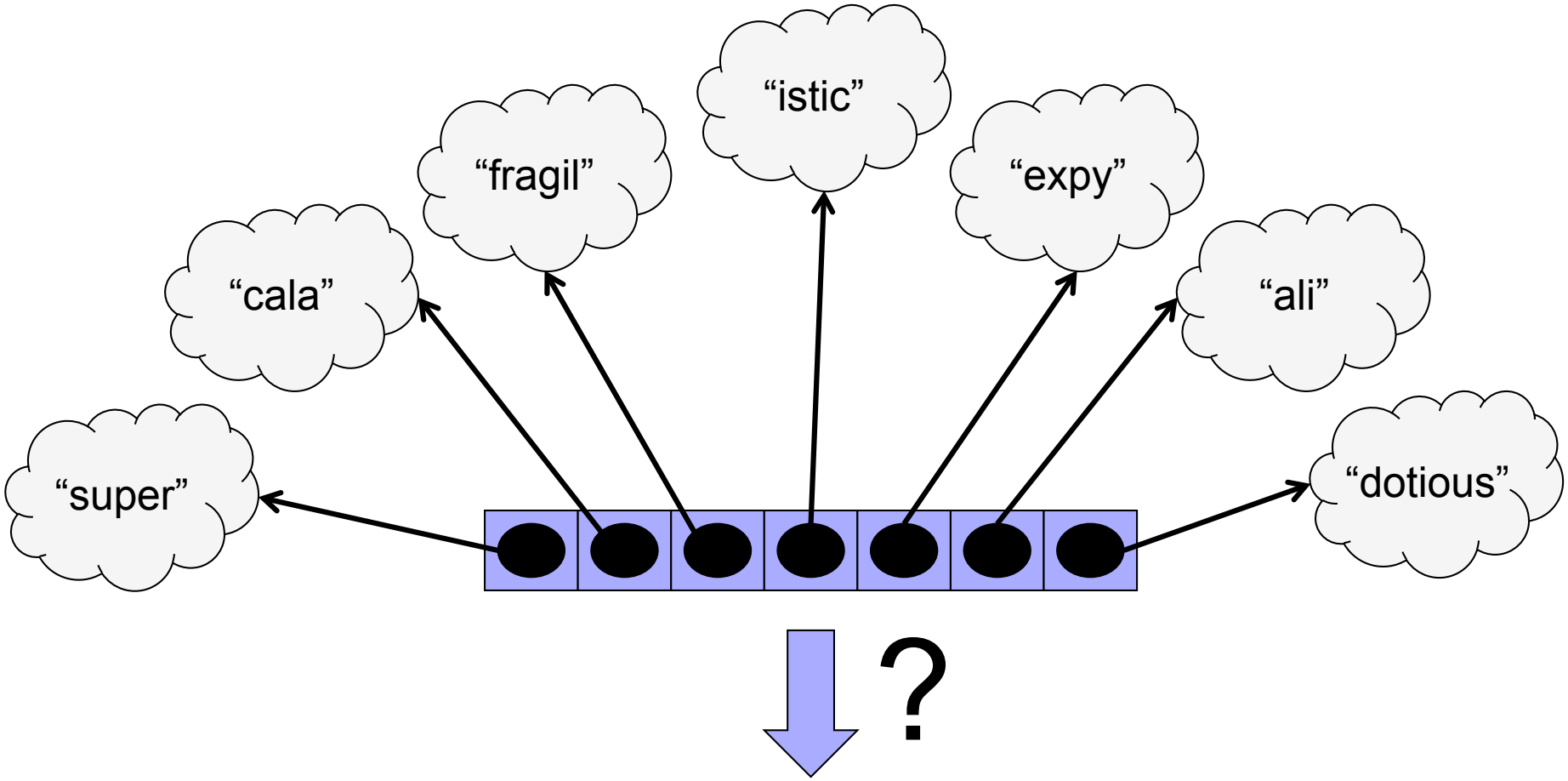
One Final Metaphor (its ‘katsconf’)



Case Study 1

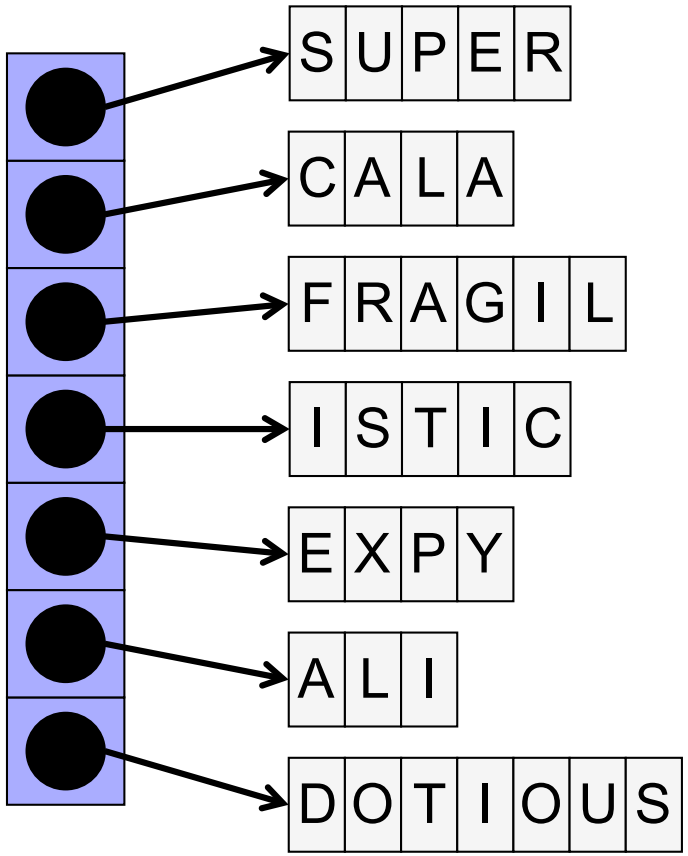
Show Me The Chars!!!!

The Problem


“istic”
“fragil”
“expy”
“cala”
“ali”
“super”
“dotious”
?
SUPERCALAFRAGILISTICEXPYALIDOTIOUS

The Theoretical Solution

flatMap(String → char [ ])



The Solution in Scala

```
object Program {
    def main(args : Array[String]) {
        val data = Array("super","cala","fragil","istic","expy","ali","dotious")
        val result = data.flatMap(_.toCharArray)

        for(c <- result) {
          printf(" %s", c);
        }
    }
}
```

```
s u p e r c a l a f r a g i l i s t i c e x p y a l i d o t i o u s
```

The Solution in Clojure

```
(let [data (vector "super" "cali" "fragil" "istic" "expy" "ali" "dotious")]
    (for [c (mapcat seq data)]
        (print c)))
```

```
s u p e r c a l a f r a g i l i s t i c e x p y a l i d o t i o u s
```

The Solution in Java (Part 1)

```
import static java.util.Arrays.*;

public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};
        data.flatMap(s -> null);
    }
}
```

The Solution in Java (Part 2)

```
import static java.util.Arrays.*;

public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};
        asList(data).flatMap(s -> null);
    }
}
```

The Solution in Java (Part 3)

```
import static java.util.Arrays.*;

public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};
        asList(data).stream().flatMap(s -> null);
    }
}
```

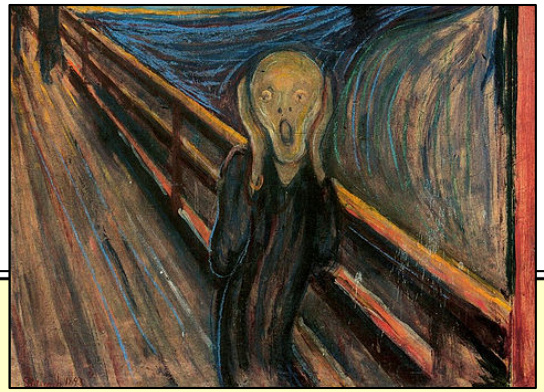
The Solution in Java (Part 4)

```
asList(data).stream().flatMap(s -> s.toCharArray());
```

```
demo1;

static java.util.Arrays.*

class Program {
lic static void main(Stri
 String [] data = {"super
 asList(data).stream().flatMap(s -> s.toCharArray());
```

```
incompatible types: cannot infer type-variable(s) R
  (argument mismatch; bad return type in lambda expression
    char[] cannot be converted to Stream<? extends R>)
 where R,T are type-variables:
  R extends Object declared in method <R>flatMap(Function<? super T,? extends Stream<? extends R>>)
  T extends Object declared in interface Stream

The type of <R>flatMap(Function<? super T,? extends Stream<? extends R>>) is erroneous
 where R,T are type-variables:
  R extends Object declared in method <R>flatMap(Function<? super T,? extends Stream<? extends R>>)
  T extends Object declared in interface Stream
----
(Alt-Enter shows hints)
```

The Solution in Java (Part 4)



```
incompatible types: cannot infer type-variable(s) R
   (argument mismatch; bad return type in lambda expression
     char[] cannot be converted to Stream<? extends R>)
  where R,T are type-variables:
    R extends Object declared in method <R>flatMap(Function<? super T,? extends Stream<? extends R>>)
    T extends Object declared in interface Stream

The type of <R>flatMap(Function<? super T,? extends Stream<? extends R>>) is erroneous
  where R,T are type-variables:
    R extends Object declared in method <R>flatMap(Function<? super T,? extends Stream<? extends R>>)
    T extends Object declared in interface Stream
----
(Alt-Enter shows hints)
```

The Solution in Java (Part 5)

```
Stream<char[]> results = asList(data)
                                .stream()
                                .flatMap(s -> asList(s.toCharArray()).stream());

for(Object obj : results.toArray()) {
    System.out.printf("%s ",obj);
}
```

```
[C@5caf905d [C@27716f4 [C@8efb846 [C@2a84aee7 [C@a09ee92 [C@30f39991 [C@452b3a41
```

The Solution in Java (Part 5)

static **DoubleStream**	**stream**(double[] array) Returns a sequential **DoubleStream** with the specified array as its source.
static **DoubleStream**	**stream**(double[] array, int startInclusive, int endExclusive) Returns a sequential **DoubleStream** with the specified range of the specified array as its source.
static **IntStream**	**stream**(int[] array) Returns a sequential **IntStream** with the specified array as its source.
static **IntStream**	**stream**(int[] array, int startInclusive, int endExclusive) Returns a sequential **IntStream** with the specified range of the specified array as its source.
static **LongStream**	**stream**(long[] array) Returns a sequential **LongStream** with the specified array as its source.
static **LongStream**	**stream**(long[] array, int startInclusive, int endExclusive) Returns a sequential **LongStream** with the specified range of the specified array as its source.
static <T> **Stream**<T>	**stream**(T[] array) Returns a sequential **Stream** with the specified array as its source.
static <T> **Stream**<T>	**stream**(T[] array, int startInclusive, int endExclusive) Returns a sequential **Stream** with the specified range of the specified array as its source.

The Solution in Java (Part 5)

```
Stream<char[]> results = asList(data)
                                    .stream()
                                    .flatMap(s -> stream(s.toCharArray()));
```

The Solution in Java (Part 6)

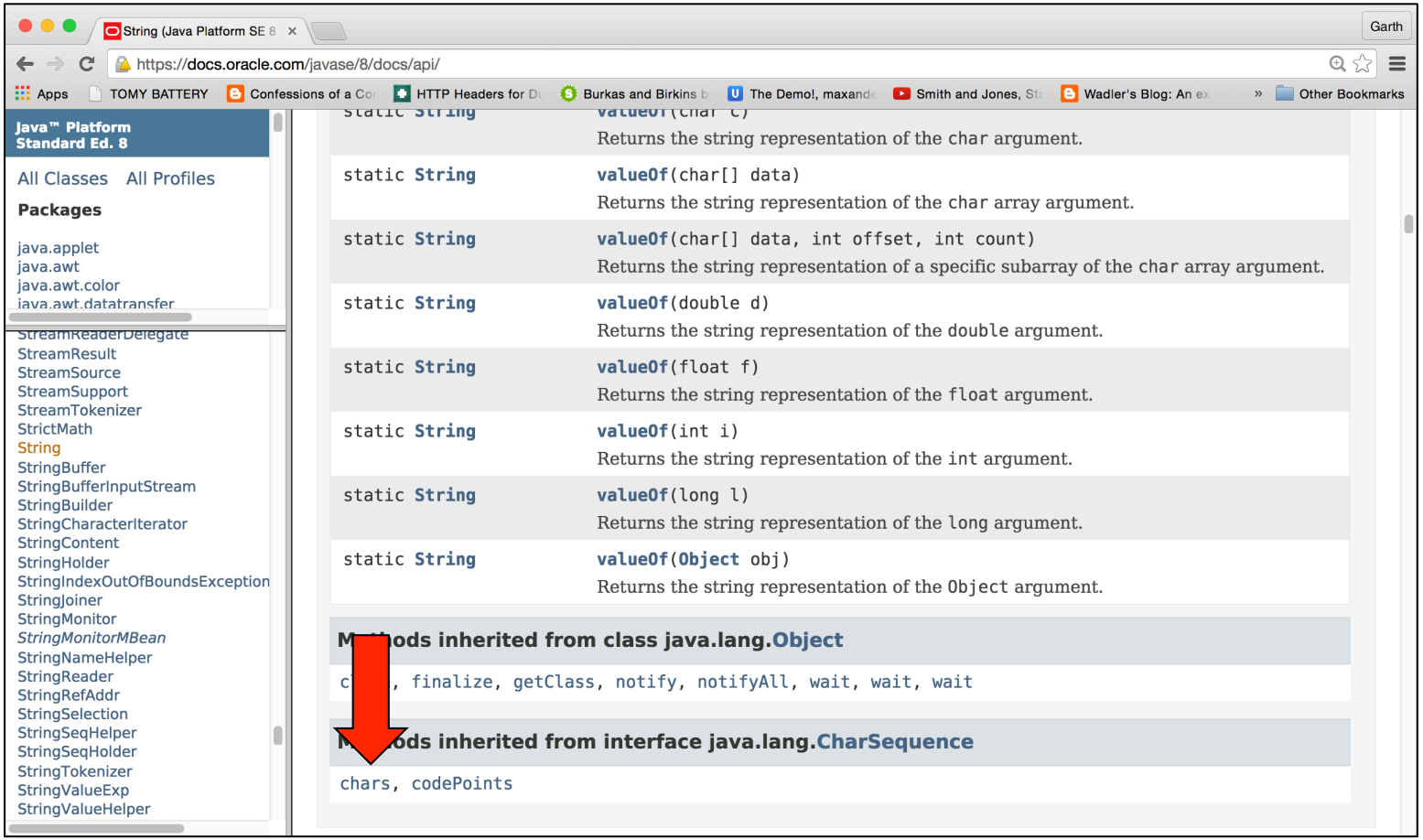
```
import java.util.Arrays;
import java.util.stream.Stream;

public class MyUtils {
    public static Stream<Character> toStream(String input) {
        Character[] output = new Character[input.length()];
        for (int i = 0; i < input.length(); i++) {
                output[i] = input.charAt(i);
        }
        return Arrays.stream(output);
    }
}
```

The Solution in Java (Part 6)

```
public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};

        Stream<Character> results = asList(data).stream()
                                                .flatMap(Program::toStream);

         for(Object obj : results.toArray()) {
                System.out.printf("%s ", obj);
        }
    }
}
```

```
s u p e r c a l a f r a g i l i s t i c e x p y a l i d o t i o u s
```

The Solution in Java (Part 7)

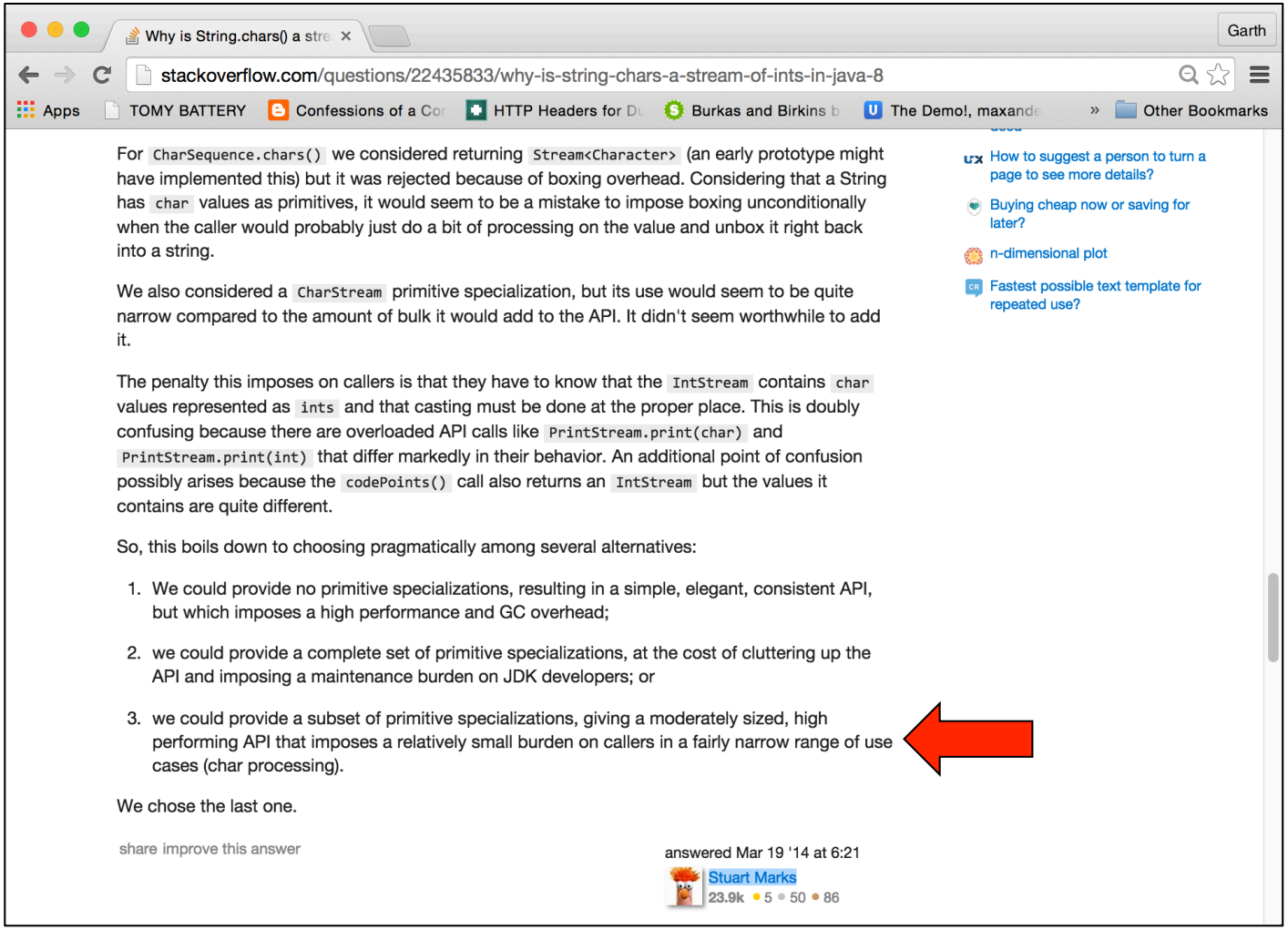


The Solution in Java (Part 7)

```
public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};

        IntStream results = asList(data).stream().flatMapToInt(s -> s.chars());

        for(int value : results.toArray()) {
            System.out.printf("%s ",(char)value);
        }
    }
}
```

s u p e r c a l a f r a g i l i s t i c e x p y a l i d o t i o u s

The Retrospective



Why is String.chars() a stre ×

Garth

stackoverflow.com/questions/22435833/why-is-string-chars-a-stream-of-ints-in-java-8

Apps | TOMY BATTERY | Confessions of a Co | HTTP Headers for D | Burkas and Birkins b | The Demo!, maxande | Other Bookmarks

For `CharSequence.chars()` we considered returning `Stream<Character>` (an early prototype might have implemented this) but it was rejected because of boxing overhead. Considering that a String has `char` values as primitives, it would seem to be a mistake to impose boxing unconditionally when the caller would probably just do a bit of processing on the value and unbox it right back into a string.

We also considered a `CharStream` primitive specialization, but its use would seem to be quite narrow compared to the amount of bulk it would add to the API. It didn't seem worthwhile to add it.

The penalty this imposes on callers is that they have to know that the `IntStream` contains `char` values represented as `ints` and that casting must be done at the proper place. This is doubly confusing because there are overloaded API calls like `PrintStream.print(char)` and `PrintStream.print(int)` that differ markedly in their behavior. An additional point of confusion possibly arises because the `codePoints()` call also returns an `IntStream` but the values it contains are quite different.

So, this boils down to choosing pragmatically among several alternatives:

1. We could provide no primitive specializations, resulting in a simple, elegant, consistent API, but which imposes a high performance and GC overhead;
2. we could provide a complete set of primitive specializations, at the cost of cluttering up the API and imposing a maintenance burden on JDK developers; or
3. we could provide a subset of primitive specializations, giving a moderately sized, high performing API that imposes a relatively small burden on callers in a fairly narrow range of use cases (char processing).

We chose the last one.

share improve this answer

answered Mar 19 '14 at 6:21

Stuart Marks

23.9k • 5 • 50 • 86

How to suggest a person to turn a page to see more details?

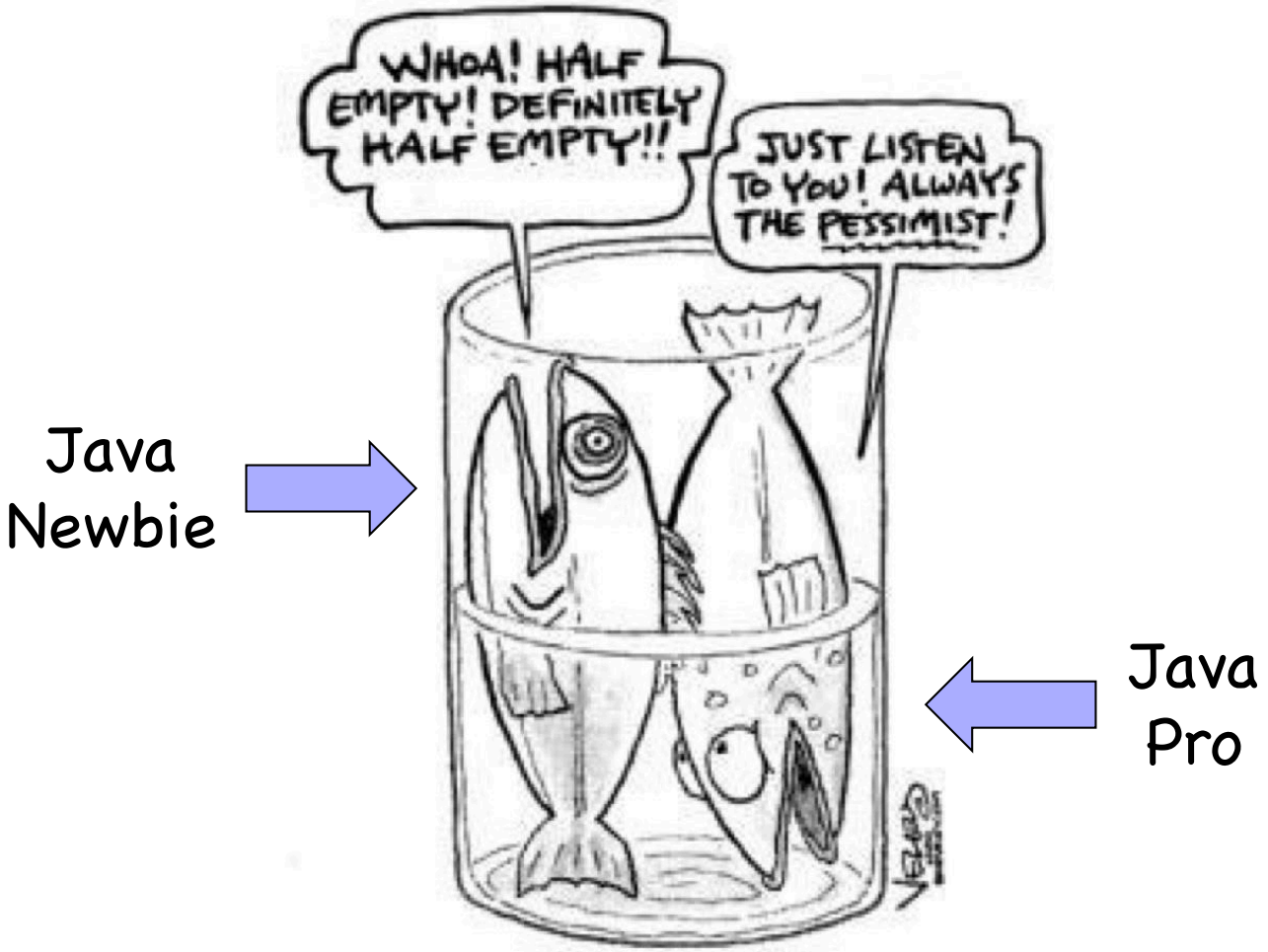
Buying cheap now or saving for later?

n-dimensional plot

Fastest possible text template for repeated use?

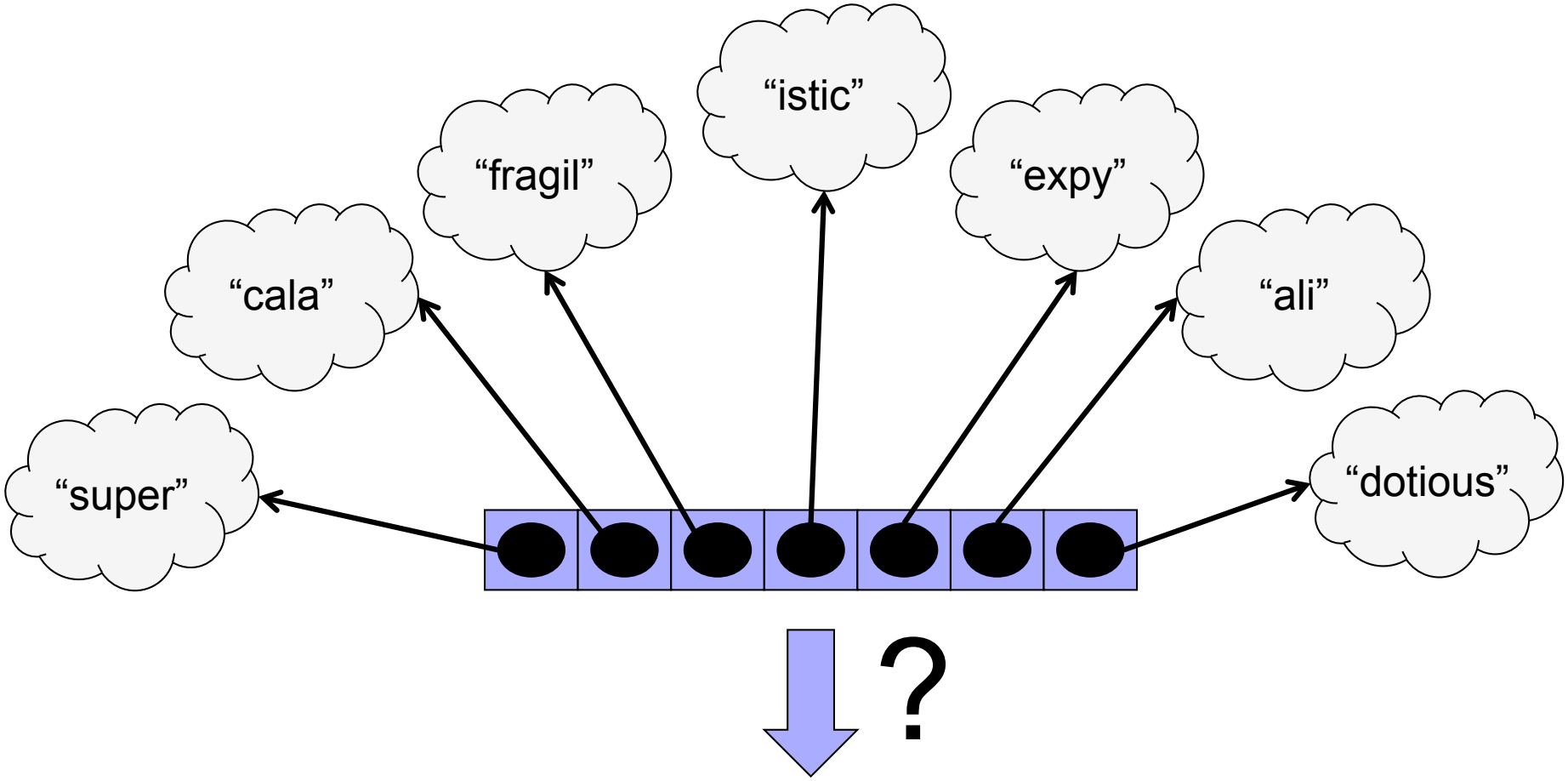
Your Verdict?

Perhaps Context Matters…



Recondite Reductions

The Problem



“supercalafragilisticexpyalidotious”

The Solution in Scala

```
object Program {
  def main(args : Array[String]): Unit = {
    val data = Array("super","cala","fragil","istic","expy","ali","dotious")

    val result = data.foldLeft(new StringBuilder())((sb,s) => sb.append(s))
    //val result = data.foldLeft(new StringBuilder())(_.append(_))

    println(result.toString)
  }
}
```

supercalafragilisticexpyalidotious

The Solution in Clojure

```
(let [data (vector "super" "cali" "fragil" "istic" "expy" "ali" "dotious")]
    (println
        (->> data
            (reduce seq)
            (apply str))))
```

or

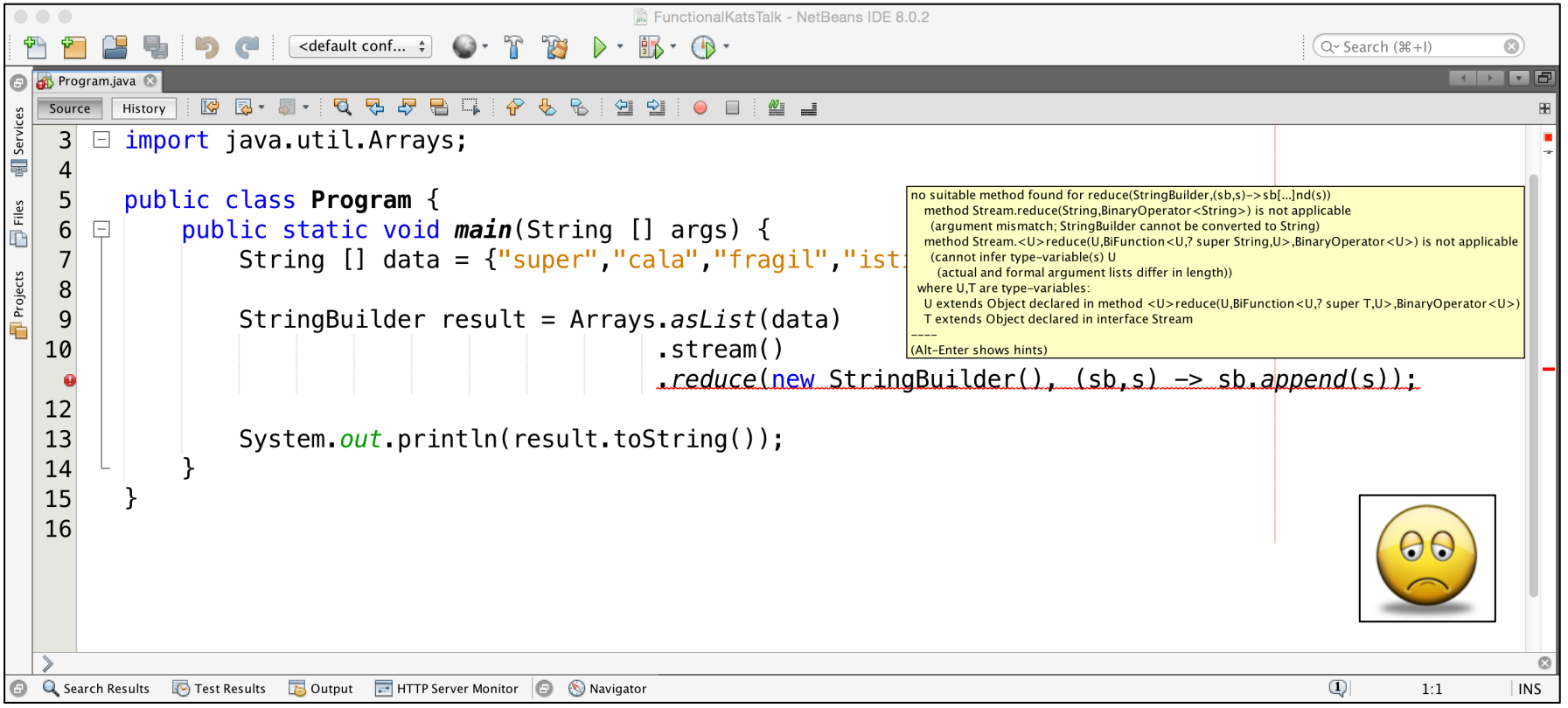
```
(let [data (vector "super" "cali" "fragil" "istic" "expy" "ali" "dotious")]
    (println
        (clojure.string/join data))
```

supercalafragilisticexpyalidotious

The Solution in Java (Part 1)

```
import java.util.Arrays;

public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};

        StringBuilder result = Arrays.asList(data)
                                     .stream()
                                     .reduce(new StringBuilder(), (sb,s) -> sb.append(s));

        System.out.println(result.toString());
    }
}
```

The Solution in Java (Part 1)



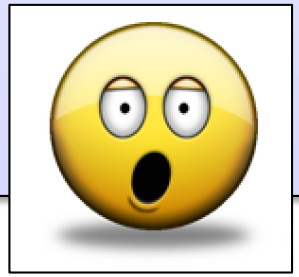
The Solution in Java (Part 2)

```
public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};

        StringBuilder result = Arrays.asList(data)
                                     .stream()
                                     .reduce(new StringBuilder(),
                                             (sb,s) -> sb.append(s),
                                             (a,b) -> new StringBuilder("wibble"));

        System.out.println(result.toString());
    }
}
```

supercalafragilisticexpyalidotious

Explaining the Solution

```
public class Program {
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};

        StringBuilder result = Arrays.asList(data)
                                     .stream()
                                     .parallel()
                                     .reduce(new StringBuilder(),
                                             (sb,s) -> sb.append(s),
                                             (a,b) -> new StringBuilder("wibble"));

        System.out.println(result.toString());
    }
}
```



wibble

Explaining the Solution

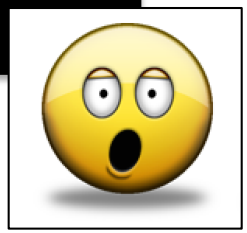
```
public class Program {
    private static StringBuffer foo(StringBuffer sb, String s) {
        String msg = "Appending %s on thread %d\n";
        long id = Thread.currentThread().getId();
        System.out.printf(msg, s, id);

        return sb.append(s);
    }
    private static StringBuffer bar(StringBuffer sb1, StringBuffer sb2) {
        String msg = "Combining function called on thread %d\n";
        long id = Thread.currentThread().getId();
        System.out.printf(msg, id);

        return sb1;
    }
```

Explaining the Solution

```
    public static void main(String [] args) {
        String [] data = {"super","cala","fragil","istic","expy","ali","dotious"};

        StringBuffer result = Arrays.asList(data)
                                    .stream()
                                    .parallel()
                                    .reduce(new StringBuffer(),
                                            Program::foo,
                                            Program::bar);
        System.out.println(result);
    }
}
```

Explaining the Solution

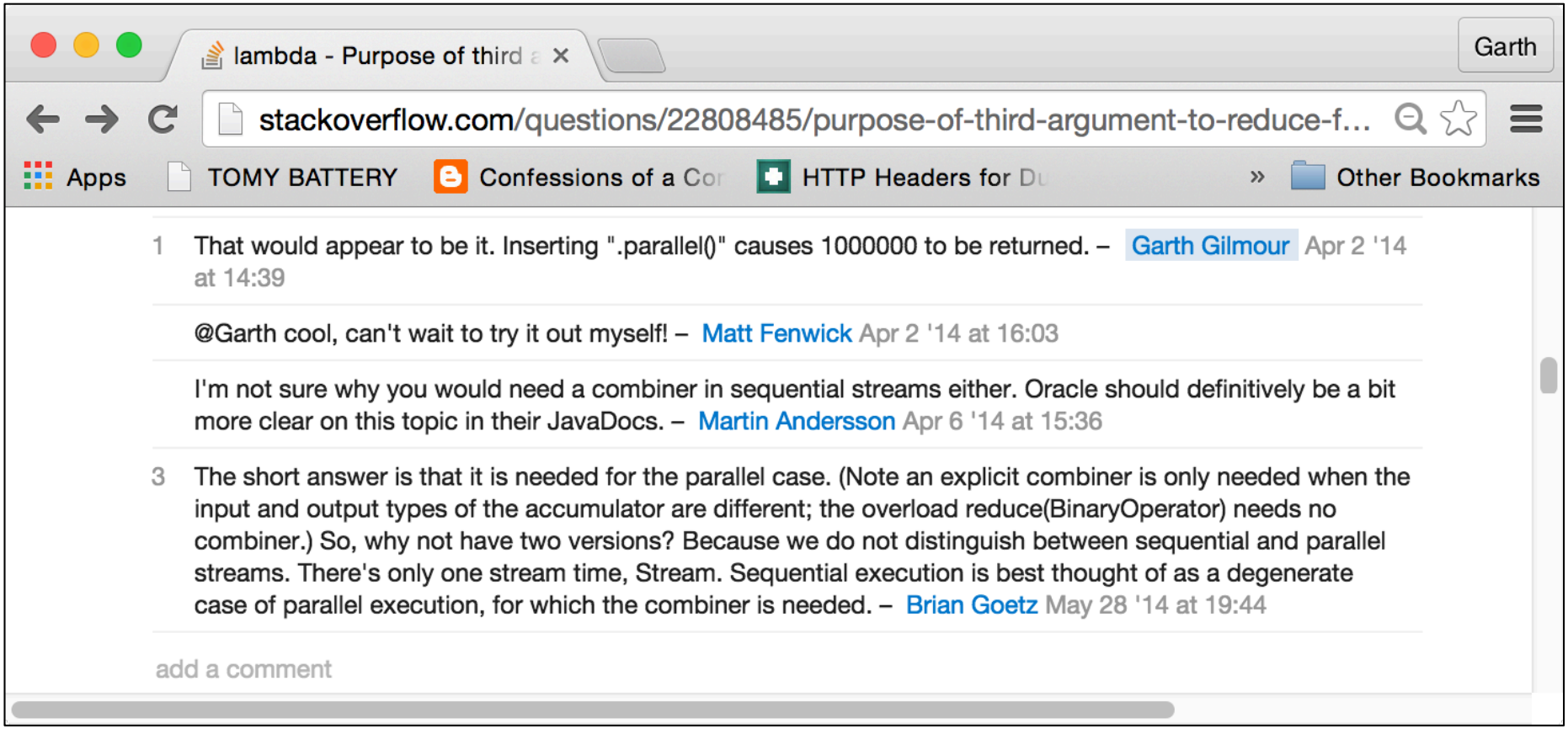
```
Appending super on thread 12
Appending istic on thread 15
Appending ali on thread 13
Appending cala on thread 10
Appending fragil on thread 14
Appending dotious on thread 11
Appending expy on thread 1
Combining function called on thread 1
Combining function called on thread 11
Combining function called on thread 11
Combining function called on thread 14
Combining function called on thread 14
Combining function called on thread 14
superisticalicalafragildotiousexpy
```



Explaining the Solution

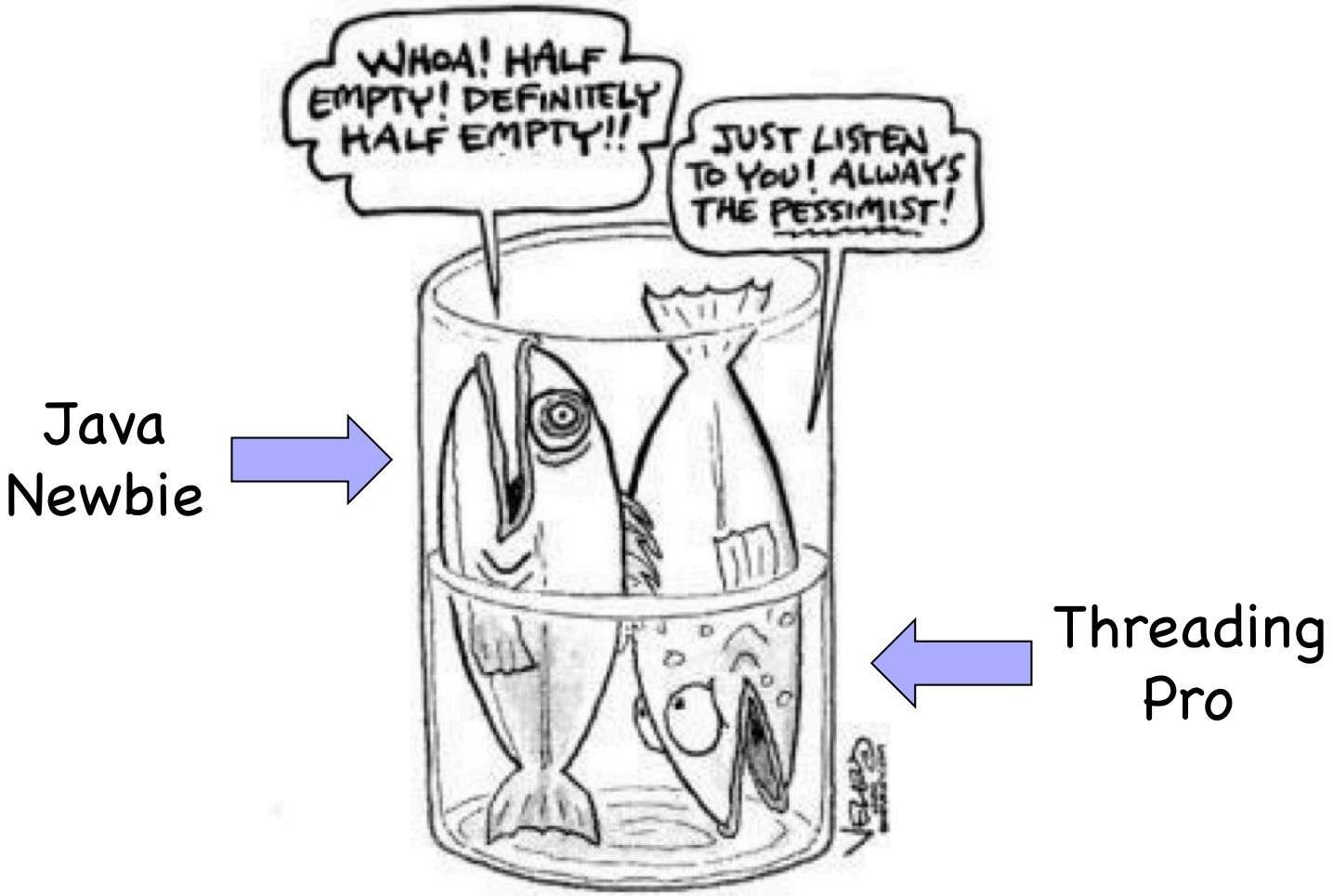
Some people, when confronted with a problem, think "I know, I'll use multi-threading".
Nothhw tpe yawrve o oblems.

The Retrospective



Your Verdict?

Perhaps Context Matters…



Lashings of Lambdas

The Problem

- We are required to:
  - Execute jobs of work on behalf of clients
  - Wrap the output of those jobs in markup
    - Where the language is a superset of HTML5

```
Function Wibble(Thing thing) Returns String
    …
End

Function newFunction = BuildMe(“h1”, Wibble)
Thing t = Foo()

//This should output the result of ‘Wibble(t)’ in ‘H1’ tags
Print(newFunction(t))
```

The Scala Solution (Plain)

```
object Program {
  def buildTaskWrappedWithTags[T](tagName : String, job : (T) => String) = {
    val openingTag = "<" + tagName + ">"
    val closingTag = "</" + tagName + ">"

    (input: T) => openingTag + job(input) +  closingTag
  }
  def main(args: Array[String]): Unit = {
    def testFuncOne(s : String)  = s.toUpperCase
    def testFuncTwo(f : File) =  f.getAbsolutePath

    val f1 = buildTaskWrappedWithTags("h1", testFuncOne)
    val f2 = buildTaskWrappedWithTags("h2", testFuncTwo)

    println(f1("wibble"))
    println(f2(new File(".")))
  }
}
```

The Scala Solution (Idiomatic)

```
object ProgramV2 {
  def buildTaskWrappedWithTags[T](tagName : String, job : (T) => String)(input : T) = {
    val openingTag = "<" + tagName + ">"
    val closingTag = "</" + tagName + ">"

    openingTag + job(input) +  closingTag
  }
  def main(args: Array[String]): Unit = {
    def testFuncOne(s : String)  = s.toUpperCase
    def testFuncTwo(f : File) =  f.getAbsolutePath

    val f1 = buildTaskWrappedWithTags("h1", testFuncOne) _
    val f2 = buildTaskWrappedWithTags("h2", testFuncTwo) _

    println(f1("wibble"))
    println(f2(new File(".")))
  }
}
```

The Clojure Solution

```
(defn buildTaskWrappedWithTags [tagName job]
    (let [openTag (str "<" tagName ">”)
          closeTag (str "</" tagName ">")]
       #(str openTag (job %) closeTag)))

(def f1 (buildTaskWrappedWithTags "h1" clojure.string/upper-case))
(def f2 (buildTaskWrappedWithTags "h2" #(.getAbsolutePath %)))

(println (f1 "big news"))
(println (f2 (clojure.java.io/file ".")))
```

Infodump - Functional Interfaces

- Most of the functional interfaces have default methods
  - E.g. ‘Consumer’ has ‘andThen’ as well as ‘accept’
- Extra versions of the interfaces exist for primitive types
  - E.g. ‘IntConsumer’, ‘LongConsumer’ & ‘DoubleConsumer’

Name	Description
Consumer<T>	Accepts a T and performs operation(s) on it
Supplier<T>	Generates a T
Predicate<T>	Accepts a T and returns a boolean
UnaryOperator<T>	Both takes and returns a T
Function<T,U>	Accepts a T and returns a U
BiFunction<T, U, V>	Takes a T and a U and returns a V

```
public class Program {
   private static <T> Function<T, String> buildTaskWrappedWithTags
                                                    (String tagName, Function<T, String> job) {
      String openingTag = "<" + tagName + ">";
      String closingTag = "</" + tagName + ">";

      return (T input) -> openingTag + job.apply(input) + closingTag;
   }
   private static String testFuncOne(String s) {
      return s.toUpperCase();
   }
   private static String testFuncTwo(File f) {
      return f.getAbsolutePath();
   }
   public static void main(String[] args) {
      Function<String, String> f1 = buildTaskWrappedWithTags("h1", Program::testFuncOne);
      Function<File, String> f2 = buildTaskWrappedWithTags("h2", Program::testFuncTwo);

      System.out.println(f1.apply("wibble"));
      System.out.println(f2.apply(new File(".")));
   }
}
```

Your Verdict?

Case Study 4

Options on Optional

Comparing Option and Optional

```
object Program {
  def fetchSystemProperty(name: String) = {
    val result = System.getProperty(name)
    if (result eq null) None else Some(result)
  }
  def main(args: Array[String]) {
    println(fetchSystemProperty("java.vendor") getOrElse "No Such Property!")
    println(fetchSystemProperty("java.version") getOrElse "No Such Property!")
    println(fetchSystemProperty("wibble") getOrElse "No Such Property!")
  }
}
```

```
Oracle Corporation
1.8.0
No Such Property!
```

Comparing Option and Optional

```
public class Program2 {
  private static Optional<String> fetchSystemProperty(String name) {
    String result = System.getProperty(name);
    if(result == null) {
        return Optional.empty();
    } else {
        return Optional.of(result);
    }
  }
  public static void main(String [] args) {
      System.out.println(fetchSystemProperty("java.vendor").orElse("No such property!"));
      System.out.println(fetchSystemProperty("java.version").orElse("No such property!"));
      System.out.println(fetchSystemProperty("wibble").orElse("No such property!"));
  }
}
```

```
Oracle Corporation
1.8.0
No Such Property!
```

The Option Monad

```
class Postcode(val value: String) {
  def findValue() = if (value eq null) None else Some(value)
}

class Address(val number: Int,
              val street: String,
              val postcode: Postcode) {
  def findPostcode() = if (postcode eq null) None else Some(postcode)
}

class Person(val name: String,
             val address: Address) {
  def findAddress() = if (address eq null) None else Some(address)
}
```

The Optional Monad

```
object Program {
  def main(args: Array[String]) {
    val p1 = new Person("Dave", null)
    val p2 = new Person("Jane",
                        new Address(10, "Arcatia Road",
                                    null))
    val p3 = new Person("Pete",
                        new Address(11, "Arcatia Road",
                                    new Postcode(null)))
    val p4 = new Person("Susan",
                        new Address(12, "Arcatia Road",
                                    new Postcode("ABC 123")))
    printPostcodeIfExists(p1)
    printPostcodeIfExists(p2)
    printPostcodeIfExists(p3)
    printPostcodeIfExists(p4)
  }
```

The Optional Monad

```
  def printPostcodeIfExists(person: Person) {
    println("Working with " + person.name)
    for (
      place <- person findAddress;
      code <- place findPostcode;
      result <- code findValue
    ) println(result)
  }
}
```

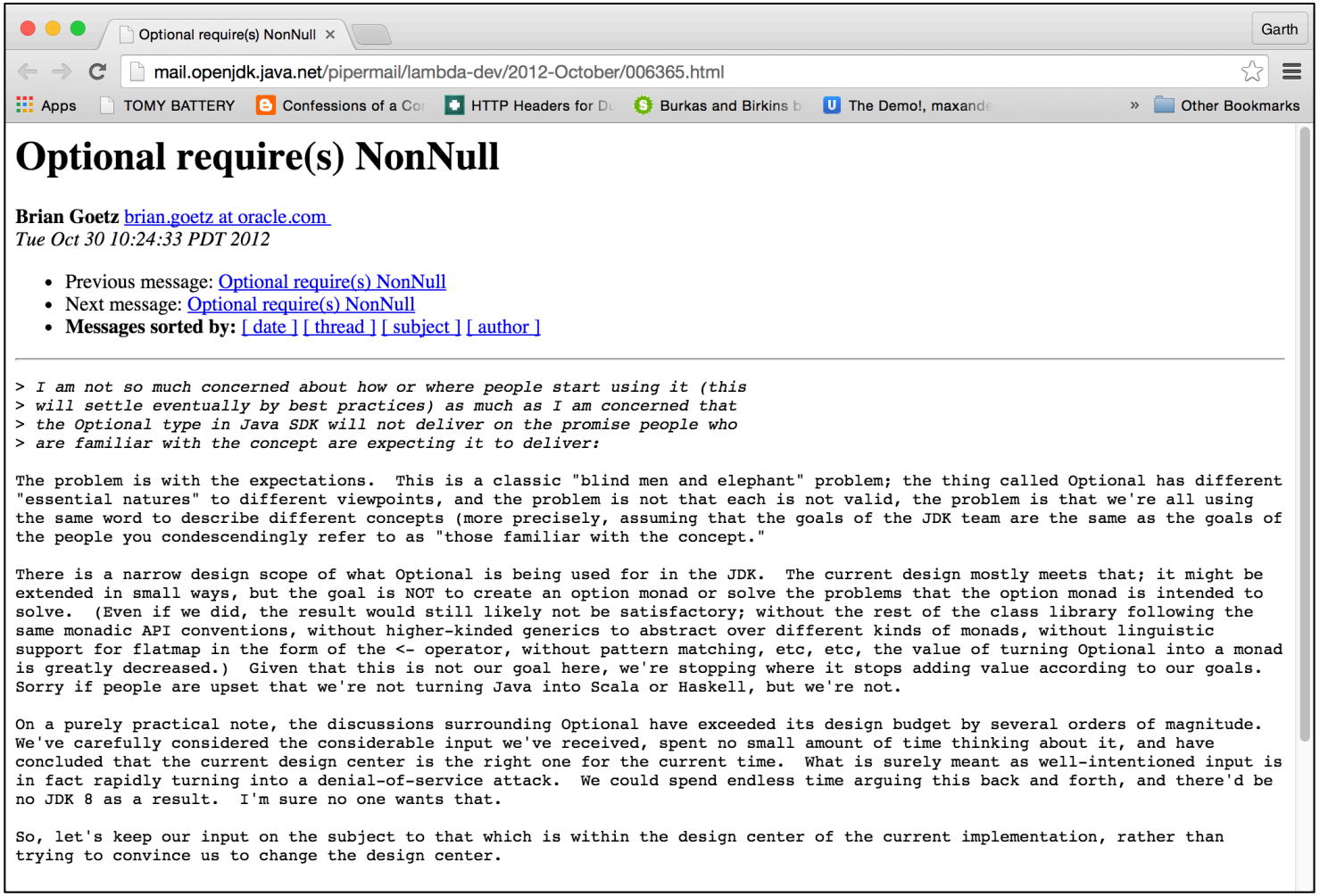
```
Working with Dave
Working with Jane
Working with Pete
Working with Susan
ABC 123
```

Option in Clojure

- Clojure doesn't really need an Option type because:
  - It is a dynamically typed language
  - Nil is handled gracefully
    - (cons "a" nil) -> ("a")
    - (count nil) -> 0
- Although we can fake it…

```
(defn fetchSystemProperty [name]
    (when-let [value (System/getProperty name)]
        value))

(or (fetchSystemProperty "java.vendor") "No such property")
(or (fetchSystemProperty "java.version") "No such property")
(or (fetchSystemProperty "java.wibble") "No such property")
```

The Retrospective



Optional require(s) NonNull

Brian Goetz brian.goetz at oracle.com
Tue Oct 30 10:24:33 PDT 2012

- Previous message: Optional require(s) NonNull
- Next message: Optional require(s) NonNull
- **Messages sorted by:** [date] [thread] [subject] [author]

> I am not so much concerned about how or where people start using it (this
> will settle eventually by best practices) as much as I am concerned that
> the Optional type in Java SDK will not deliver on the promise people who
> are familiar with the concept are expecting it to deliver:

The problem is with the expectations. This is a classic "blind men and elephant" problem; the thing called Optional has different "essential natures" to different viewpoints, and the problem is not that each is not valid, the problem is that we're all using the same word to describe different concepts (more precisely, assuming that the goals of the JDK team are the same as the goals of the people you condescendingly refer to as "those familiar with the concept."

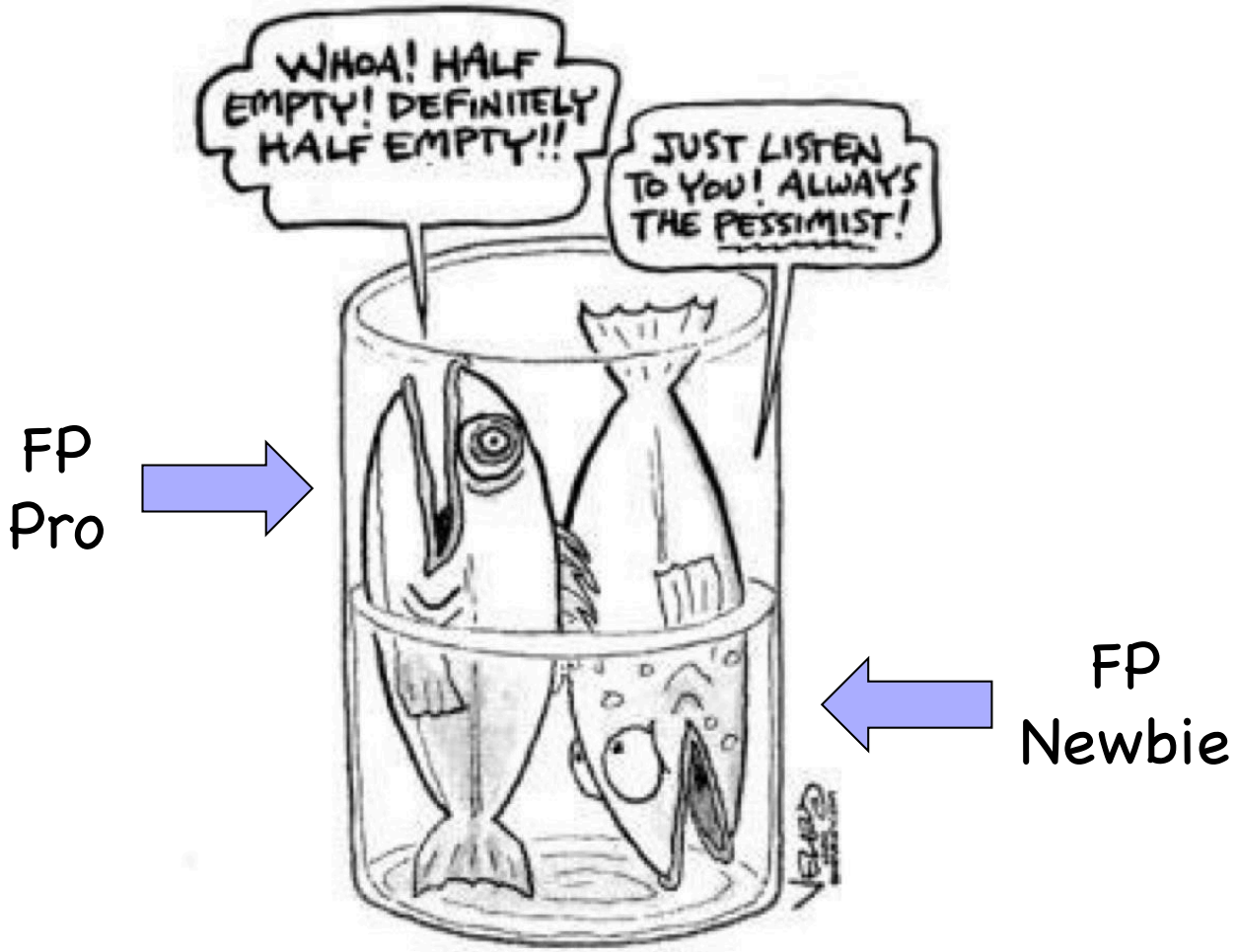
There is a narrow design scope of what Optional is being used for in the JDK. The current design mostly meets that; it might be extended in small ways, but the goal is NOT to create an option monad or solve the problems that the option monad is intended to solve. (Even if we did, the result would still likely not be satisfactory; without the rest of the class library following the same monadic API conventions, without higher-kinded generics to abstract over different kinds of monads, without linguistic support for flatmap in the form of the <- operator, without pattern matching, etc, etc, the value of turning Optional into a monad is greatly decreased.) Given that this is not our goal here, we're stopping where it stops adding value according to our goals. Sorry if people are upset that we're not turning Java into Scala or Haskell, but we're not.

On a purely practical note, the discussions surrounding Optional have exceeded its design budget by several orders of magnitude. We've carefully considered the considerable input we've received, spent no small amount of time thinking about it, and have concluded that the current design center is the right one for the current time. What is surely meant as well-intentioned input is in fact rapidly turning into a denial-of-service attack. We could spend endless time arguing this back and forth, and there'd be no JDK 8 as a result. I'm sure no one wants that.

So, let's keep our input on the subject to that which is within the design center of the current implementation, rather than trying to convince us to change the design center.

Your Verdict?

Perhaps Context Matters…

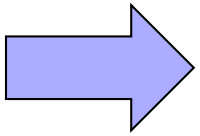


Case Study 5

Counting Cards

The Problem

3H JS 3C 7C 5D	Pair
JH 2C JD 2H 4C	Two Pair
5H 9H 6H 7H 8H	Straight Flush
9H 9D 3S 9S 9C	Four Of A Kind
9C 3H 9S 9H 3S	Full House
3C 6C 9C 2C 9C	Flush
3C 3H 9C 2D 9H	Two Pair
5C 8C 7D 9D 6S	Straight
KH 10H AH JH QH	Royal Flush
7C 2H KD 7H 7S	Three Of A Kind
2C 2H KD 7H 7S	Two Pair
1C 2D 3C 4D 6S	Highest Card



Methods of the ‘Functional Toolkit’

```
class PokerHand(hand: Seq[Card]) {
  def name = {
    if (isRoyalFlush)              "Royal Flush"
    else if(isStraightFlush)       "Straight Flush"
    else if(isFourOfAKind)         "Four Of A Kind"
    else if(isFullHouse)           "Full House"
    else if(isFlush)               "Flush"
    else if(isStraight)            "Straight"
    else if(isThreeOfAKind)        "Three Of A Kind"
    else if(isTwoPair)             "Two Pair"
    else if(isPair)                "Pair"
    else                            "Highest Card"
  }
  override def toString = hand.foldLeft("")(_ + " " + _)

  def nOf(count: Int)(rank : String) = hand.count(_.rank == rank) >= count
  def twoOf = nOf(2)_
  def threeOf = nOf(3)_
  def fourOf = nOf(4)_
```

```
  def isFlush                    = allOfSameSuit
  def isRoyalFlush               = allOfSameSuit && allCardsHigh
  def isStraightFlush            = isStraight && allOfSameSuit
  def isFourOfAKind              = allRanks.exists(fourOf(_))
  def isThreeOfAKind             = allRanks.exists(threeOf(_))
  def isTwoPair                  = allRanks.combinations(2)
                                          .exists(pair => twoOf(pair(0)) && twoOf(pair(1)))
  def isPair                     = allRanks.exists(twoOf(_))
  def isStraight                 = allRanks.containsSlice(ranksInDescendingOrder)
  def isFullHouse = {
     def twoOfThenThreeOf(pair : List[String]) = twoOf(pair(0)) && threeOf(pair(1))
     def threeOfThenTwoOf(pair : List[String]) = threeOf(pair(0)) && twoOf(pair(1))

     allRanks.combinations(2)
             .exists(pair => twoOfThenThreeOf(pair) || threeOfThenTwoOf(pair))
  }
  def allOfSameSuit = hand.forall(_.suit == hand(0).suit)
  def allCardsHigh =  Array("A","Q","K","J","10").forall(r => hand.exists(_.rank == r))
  def ranksInDescendingOrder = hand.map(_.rank)
                                   .sortWith(allRanks.indexOf(_) < allRanks.indexOf(_))
}
```

Methods Used In Unit Tests

```
abstract class PokerSpec extends FlatSpec with Matchers {
  def handWorks(cards : String, identifier: (PokerHand) => Boolean) {
    val hand = PokerHand.buildHand(cards)
    identifier(hand) should be (true)
  }
  def worksInAnyOrder(cards : List[String], func: (PokerHand) => Boolean) {
    for(cards <- cards.permutations.map(_.mkString(" "))) {
      handWorks(cards, func)
    }
  }
}
```

```
class FourOfAKindSpec extends PokerSpec {
  "Four of a kind " should "be recognised" in {
    handWorks("8S 8D 8C 8H 4S", _.isFourOfAKind)
  }
  it should "be recognised regardless of order" in {
    val cards = List("8S", "8D", "8C", "8H", "4S")
    worksInAnyOrder(cards, _.isFourOfAKind)
  }
}
```

Methods Used In Unit Tests

```
abstract class PokerSpec extends FlatSpec with Matchers {
  def handWorks(cards : String, identifier: (PokerHand) => Boolean) {
    val hand = PokerHand.buildHand(cards)
    identifier(hand) should be (true)
  }
  def worksInAnyOrder(cards : List[String], func: (PokerHand) => Boolean) {
    for(cards <- cards.permutations.map(_.mkString(" "))) {
      handWorks(cards, func)
    }
  }
}
```

```
class FourOfAKindSpec extends PokerSpec {
  "Four of a kind " should "be recognised" in {
    handWorks("8S 8D 8C 8H 4S", _.isFourOfAKind)
  }
  it should "be recognised regardless of order" in {
    val cards = List("8S", "8D", "8C", "8H", "4S")
    worksInAnyOrder(cards, _.isFourOfAKind)
  }
}
```

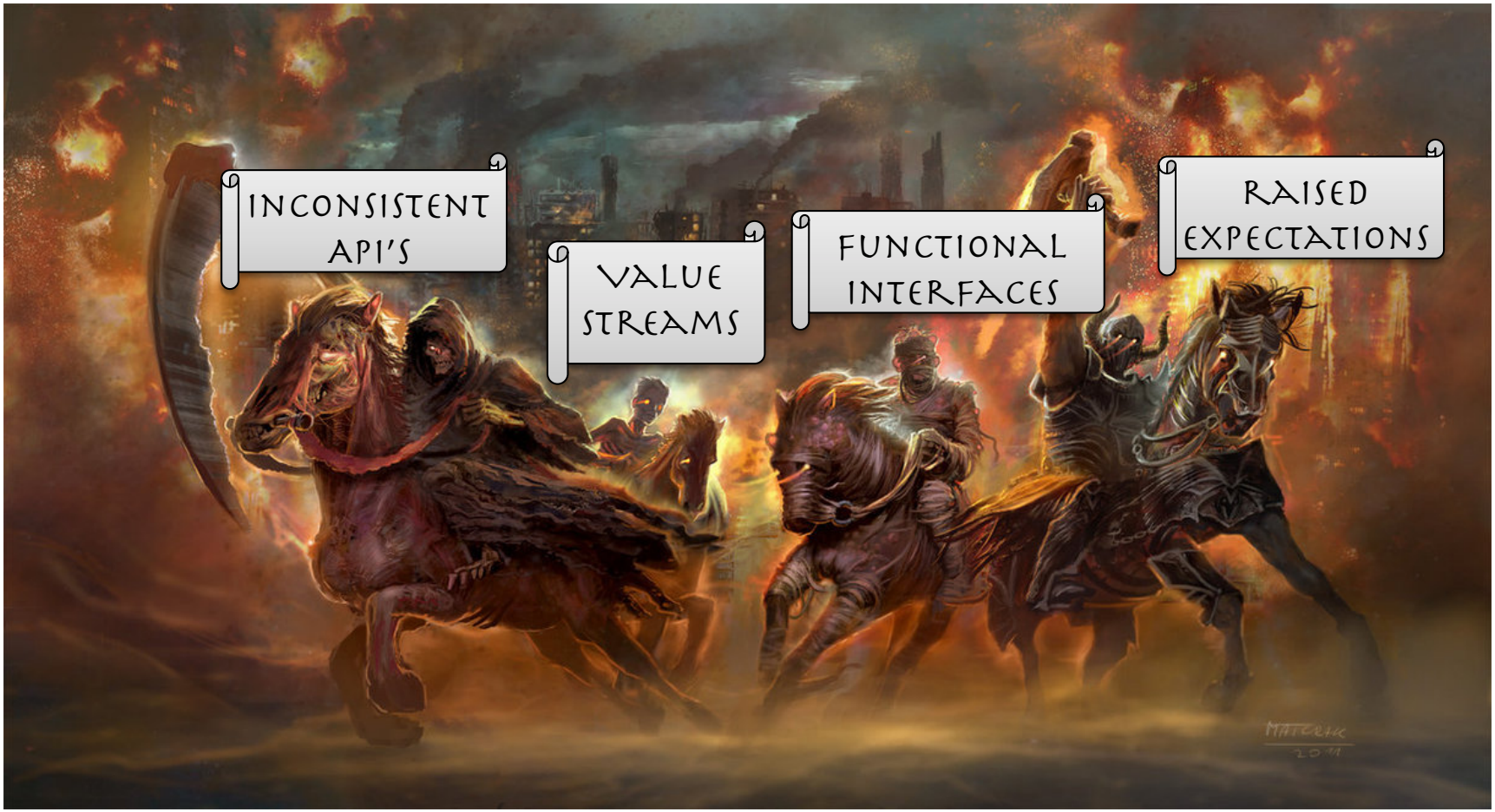
Your Verdict?

Conclusions

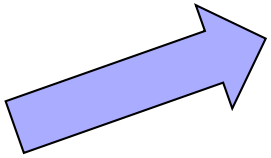
Plus Future Gazing…

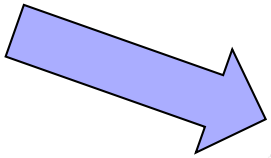
THE (OLD) HORSEMEN OF THE JAVAPOCALYPSE...

THE (NEW) HORSEMEN OF THE JAVAPOCALYPSE...

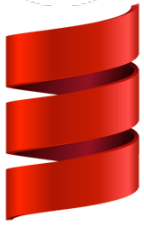


Your Verdict?





Clojure



Scala