

# Java.next

QConSF, November 2011

Erik Onnen

# About Me

- Director of Architecture at Urban Airship (1 year)
- Previously Principal Engineer at Jive Software (3 years)
- 13 years writing Java, Python, C++
- Decent amount of hacking in Scala, Clojure, Ruby



# Caveats



# Caveats

- I like lots of things about Scala, Clojure and Groovy



# Caveats

- I like lots of things about Scala, Clojure and Groovy
- I **am not** a programming language student, author or expert



# Caveats

- I like lots of things about Scala, Clojure and Groovy
- I **am not** a programming language student, author or expert
- I **am** a practitioner



# Caveats

- I like lots of things about Scala, Clojure and Groovy
- I **am not** a programming language student, author or expert
- I **am** a practitioner
  - Write lots of code quickly with good tools



# Caveats

- I like lots of things about Scala, Clojure and Groovy
- I **am not** a programming language student, author or expert
- I **am** a practitioner
  - Write lots of code quickly with good tools
  - Move quickly when troubleshooting production code





# Caveats

- I like lots of things about Scala, Clojure and Groovy
- I **am not** a programming language student, author or expert
- I **am** a practitioner
  - Write lots of code quickly with good tools
  - Move quickly when troubleshooting production code
  - Move quickly when re-engaging with code



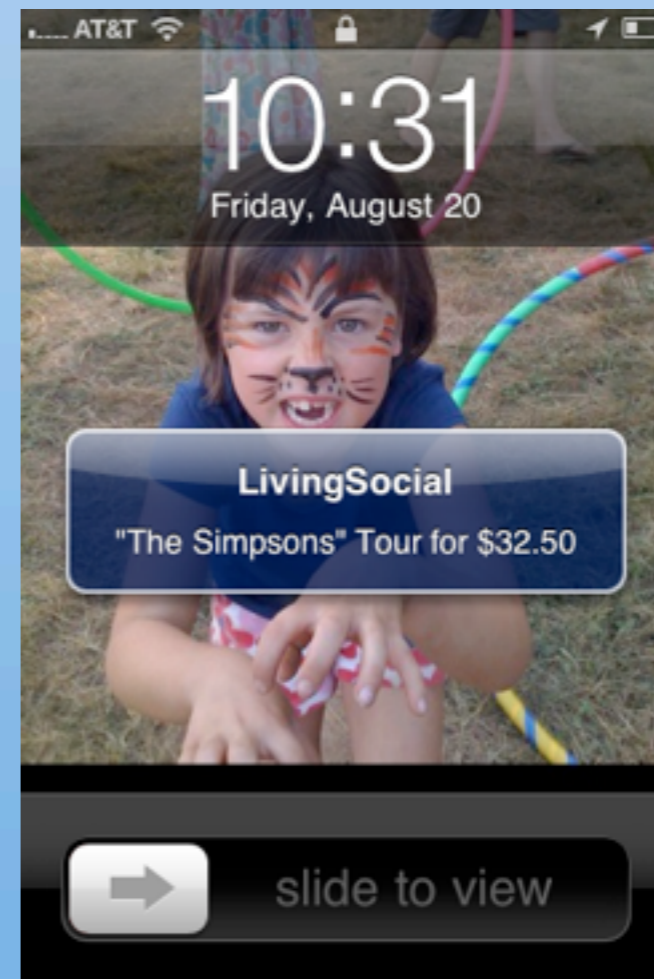
# In this Talk

- About Urban Airship
- Java at Urban Airship
- Java and the JVM
- Dat Tool
- Come at me Troll!
- What do we need to improve?
- What does the language need to improve?



# What is an Urban Airship?

- Hosting for mobile services that developers should not build themselves
- Unified API for services across platforms
- SLAs for throughput, latency



FACT



# FACT

- Over 50 Java services in production



# FACT

- Over 50 Java services in production
  - HTTP endpoints



# FACT

- Over 50 Java services in production
  - HTTP endpoints
  - Databases



# FACT

- Over 50 Java services in production
  - HTTP endpoints
  - Databases
  - Message routing and delivery





# FACT

- Over 50 Java services in production
  - HTTP endpoints
  - Databases
  - Message routing and delivery
  - Large scale socket management using NIO



# FACT

- Over 50 Java services in production
  - HTTP endpoints
  - Databases
  - Message routing and delivery
  - Large scale socket management using NIO
  - Large scale data analysis



# FACT

- Over 50 Java services in production
  - HTTP endpoints
  - Databases
  - Message routing and delivery
  - Large scale socket management using NIO
  - Large scale data analysis
- We also eschew most “Enterprise” Java



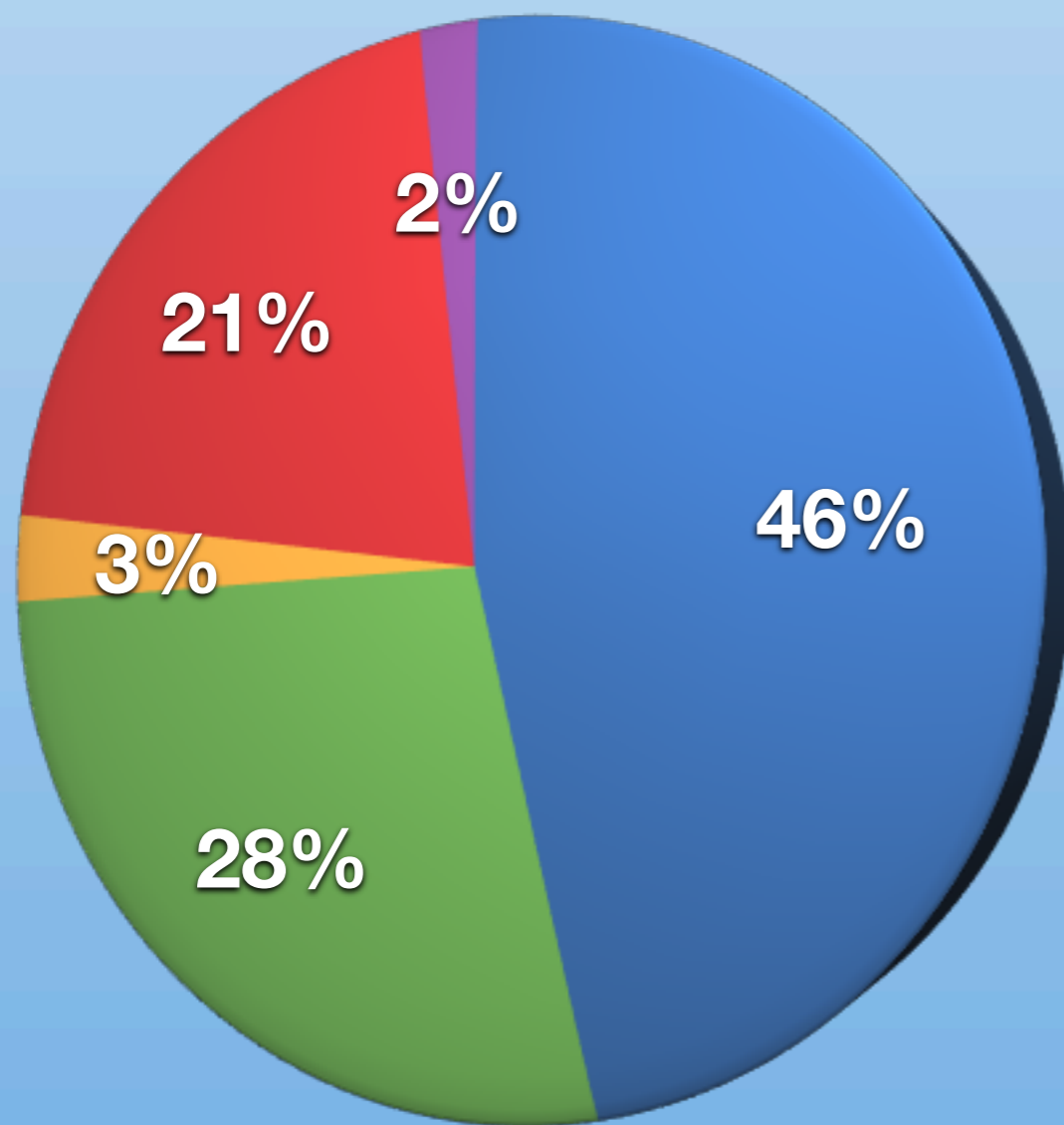
# FACT

- Over 50 Java services in production
  - HTTP endpoints
  - Databases
  - Message routing and delivery
  - Large scale socket management using NIO
  - Large scale data analysis
- We also eschew most “Enterprise” Java
- Everything in this talk we practice (and it works really well for us)



# FACT

## A Day in UA Engineering



- New Feature Development
- Sustaining Engineering
- IRC Tomfoolery
- Production Support
- Beer/Pong



# Why Use The JVM?



# Why Use The JVM?

- Fast.



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths





# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(
- Consistent, coherent memory model and concurrency - no “undefined behavior”



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(
- Consistent, coherent memory model and concurrency - no “undefined behavior”
- Scalable NIO



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(
- Consistent, coherent memory model and concurrency - no “undefined behavior”
- Scalable NIO
- Threading, with a clean signal and interrupt handling



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(
- Consistent, coherent memory model and concurrency - no “undefined behavior”
- Scalable NIO
- Threading, with a clean signal and interrupt handling
- Introspect the runtime with little to no impact



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(
- Consistent, coherent memory model and concurrency - no “undefined behavior”
- Scalable NIO
- Threading, with a clean signal and interrupt handling
- Introspect the runtime with little to no impact
- Snapshot the runtime under duress and analyze later



# Why Use The JVM?

- Fast.
  - Networking, disk I/O, maths
  - Unless the GC is running :(
- Consistent, coherent memory model and concurrency - no “undefined behavior”
- Scalable NIO
- Threading, with a clean signal and interrupt handling
- Introspect the runtime with little to no impact
- Snapshot the runtime under duress and analyze later
- But - none of these are Java-specific



# Why Java on the JVM?





# Why Java on the JVM?

- There are many hammers available



# Why Java on the JVM?

- There are many hammers available
  - All are good at something



# Why Java on the JVM?

- There are many hammers available
  - All are good at something
  - They are almost all interesting



# Why Java on the JVM?

- There are many hammers available
  - All are good at something
  - They are almost all interesting
  - Not all are OK at everything



# Why Java on the JVM?



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code
  - No multiple inheritance





# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code
  - No multiple inheritance
  - No monkey patching or trait collision rules to memorize



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code
  - No multiple inheritance
  - No monkey patching or trait collision rules to memorize
  - No hacking the global namespace or meta class munging



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code
  - No multiple inheritance
  - No monkey patching or trait collision rules to memorize
  - No hacking the global namespace or meta class munging
  - No duck typing or hidden type coercion



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code
  - No multiple inheritance
  - No monkey patching or trait collision rules to memorize
  - No hacking the global namespace or meta class munging
  - No duck typing or hidden type coercion
- Favors principle of least astonishment



# Why Java on the JVM?

- Java as a language is inherently simple - WYSIWYG (mostly)
  - Class behaviors are known by reading class code
  - No multiple inheritance
  - No monkey patching or trait collision rules to memorize
  - No hacking the global namespace or meta class munging
  - No duck typing or hidden type coercion
- Favors principle of least astonishment

*“When you specifically try to dumb down good ideas for the masses, you reveal your contempt of said masses. Case study: Java.” - David Hansson*



# Why Java on the JVM?



# Why Java on the JVM?

- Java is generally easy to read



# Why Java on the JVM?

- Java is generally easy to read
- Easy to read means easy to **maintain**





# Why Java on the JVM?

- Java is generally easy to read
- Easy to read means easy to **maintain**

*Loop Recognition in C++/Java/Go/Scala - Robert Hundt, Google*



# Why Java on the JVM?

- Java is generally easy to read
- Easy to read means easy to **maintain**

```
(1 to 10).foreach(  
  _ => {  
    n2  
    .edge  
    .repeat(100,  
      _.back(_.edge  
        .repeat(25,  
          baseLoop(_)))  
      .edge)  
    .connect(n1)  
  })
```

*Loop Recognition in C++/Java/Go/Scala - Robert Hundt, Google*



# Why Java on the JVM?

- Java is generally easy to read
- Easy to read means easy to **maintain**

```
(1 to 10).foreach(  
  _ => {  
    n2  
    .edge  
    .repeat(100,  
      _.back(_.edge  
        .repeat(25,  
          baseLoop(_)))  
    .edge)  
    .connect(n1)  
  })
```

```
public void buildConnect(int start, int end) {  
    new BasicBlockEdge(cfg, start, end);  
}  
  
public int buildDiamond(int start) {  
    int bb0 = start;  
    new BasicBlockEdge(cfg, bb0, bb0 + 1);  
    new BasicBlockEdge(cfg, bb0, bb0 + 2);  
    new BasicBlockEdge(cfg, bb0 + 1, bb0 + 3);  
    new BasicBlockEdge(cfg, bb0 + 2, bb0 + 3);  
    return bb0 + 3;  
}  
  
public int buildStraight(int start, int n) {  
    for (int i = 0; i < n; i++) {  
        buildConnect(start + i, start + i + 1);  
    }  
    return start + n;  
}  
  
// Construct a simple loop with two diamonds in it  
public int buildBaseLoop(int from) {  
    int header = buildStraight(from, 1);  
    int diamond1 = buildDiamond(header);  
    int d11 = buildStraight(diamond1, 1);  
    int diamond2 = buildDiamond(d11);  
    int footer = buildStraight(diamond2, 1);  
    buildConnect(diamond2, d11);  
    buildConnect(diamond1, header);  
  
    buildConnect(footer, from);  
    footer = buildStraight(footer, 1);  
    return footer;  
}  
  
public static void main(String[] args) {  
    app.buildBaseLoop(0);  
}
```

# Why Java on the JVM?



# Why Java on the JVM?

- Almost no transparent performance degradation





# Why Java on the JVM?

- Almost no transparent performance degradation

```
(defn- parse-headers
  "Returns a map of the response headers from connection."
  [#^URLConnection connection]
  (let [hs (.getHeaderFields connection)]
    (into {} (for [[k v] hs :when k] [(keyword (.toLowerCase k)) (seq v)]))))

(defn- parse-cookies
  "Returns a map of cookies when given the Set-Cookie string sent
  by a server."
  [#^String cookie-string]
  (when cookie-string
    (into {}
      (for [#^String cookie (.split cookie-string ";")]
        (let [keyval (map (fn [#^String x] (.trim x)) (.split cookie "=" 2))]
          [(first keyval) (second keyval)]))))))

(defn- create-cookie-string
  "Returns a string suitable for sending to the server in the
  \"Cookie\" header when given a clojure map of cookies."
  [cookie-map]
  (str-join "; " (map (fn [cookie]
    (str #^String (as-str (key cookie))
      "="
      #^String (as-str (val cookie))))
    cookie-map)))
```



# Why Java on the JVM?



# Why Java on the JVM?

- No impedance with runtime introspection





# Why Java on the JVM?

- No impedance with runtime introspection
  - A thread dump is a thread dump



# Why Java on the JVM?

- No impedance with runtime introspection
  - A thread dump is a thread dump
  - A heap dump is a heap dump





# Why Java on the JVM?

	421,657	100 %
	420,521	99 %
pre.server. <b>RequestDecoder.messageReceived</b> (ChannelHandlerContext, MessageEvent)	467,476	111 %
r.core.server. <b>RequestDecoder.handleRequest</b> (ChannelHandlerContext, FramedRead)	463,152	110 %
el. <b>SimpleChannelHandler.handleUpstream</b> (ChannelHandlerContext, ChannelEvent)	427,156	101 %
reactor.core.server. <b>RequestHandler.messageReceived</b> (ChannelHandlerContext, MessageEvent)	426,750	101 %
hip.radon.proxy.rpc. <b>RadonProxyCommandHandler.doHandleRequest</b> (Channel, Reactor\$Request)	426,445	101 %
irship.radon.proxy.rpc. <b>RadonProxyCommandHandler.proxyWrite</b> (Channel, RequestTuple)	397,252	94 %
anairship.radon.proxy.routing. <b>AsyncWriteShard.write</b> (RequestTuple)	283,673	67 %
.javaapi.producer. <b>Producer.send</b> (ProducerData)	264,097	63 %
kafka.producer. <b>Producer.send</b> (Seq)	260,786	62 %
kafka.producer. <b>ProducerPool.send</b> (Seq)	172,580	41 %
scala.collection.mutable. <b>ResizableArray\$class.foreach</b> (ResizableArray, Function1)	134,009	32 %
kafka.producer. <b>ProducerPool\$\$\$anonfun\$send\$1.apply</b> (Object)	131,178	31 %
kafka.producer. <b>ProducerPool\$\$\$anonfun\$send\$1.apply\$mcVI\$sp</b> (int)	130,567	31 %
scala.collection.mutable. <b>ArrayBuffer.map</b> (Function1, CanBuildFrom)	83,327	20 %
scala.collection. <b>TraversableLike\$class.map</b> (TraversableLike, Function1, CanBuildFrom)	83,015	20 %
scala.collection.mutable. <b>ResizableArray\$class.foreach</b> (ResizableArray, Function1)	71,628	17 %
scala.collection.mutable. <b>ArrayBuffer.sizeHint</b> (TraversableLike, int)	4,435	1 %
scala.collection.generic. <b>TraversableFactory\$GenericCanBuildFrom.apply</b> (Object)	3,596	1 %
scala.collection.mutable. <b>ArrayBuffer.result</b> ()	983	0 %
scala.collection. <b>TraversableLike\$\$\$anonfun\$map\$1.&lt;init&gt;</b> (TraversableLike, Function1, Builder)	477	0 %
scala.collection.mutable. <b>ArrayBuffer.sizeHint\$default\$2</b> ()	154	0 %
scala.collection.mutable. <b>ArrayBuffer.repr</b> ()	152	0 %
kafka.producer. <b>SyncProducer.send</b> (String, int, ByteBufferMessageSet)	21,870	5 %
scala.collection.mutable. <b>ArrayBuffer.partition</b> (Function1)	14,667	3 %
scala.collection.mutable. <b>ArrayBuffer.apply</b> (int)	1,707	0 %
scala.collection.mutable. <b>StringBuilder.&lt;init&gt;</b> ()	1,590	0 %
scala.collection.mutable. <b>ArrayBuffer.size</b> ()	1,120	0 %
scala.collection. <b>Seq\$.canBuildFrom</b> ()	642	0 %
scala.collection.mutable. <b>StringBuilder.append</b> (Object)	590	0 %
kafka.producer. <b>ProducerPool\$\$\$anonfun\$send\$1\$\$\$anonfun\$3.&lt;init&gt;</b> (ProducerPool\$\$\$anonfun\$send\$1)	533	0 %

# Why Java on the JVM?



# Why Java on the JVM?

- Little idiomatic impedance with the runtime





# Why Java on the JVM?

- Little idiomatic impedance with the runtime

<http://dev.bizo.com/2010/01/scala-supports-non-local-returns.html>

# Why Java on the JVM?

- Little idiomatic impedance with the runtime

```
object Foo {  
  def main(args: Array[String]) {  
    foo(List(1, 2, 3))  
  }  
  
  def foo(l: List[Int]): Int = {  
    l.foreach { (i) =>  
      println(i)  
    }  
    return 5  
  }  
}
```

<http://dev.bizo.com/2010/01/scala-supports-non-local-returns.html>

# Why Java on the JVM?





# Why Java on the JVM?

- The tools



# Why Java on the JVM?

- The tools
  - People have mixed reactions to refactoring tools



# Why Java on the JVM?

- The tools
  - People have mixed reactions to refactoring tools
  - IDEs starting to “learn”



# Why Java on the JVM?

- The tools
  - People have mixed reactions to refactoring tools
  - IDEs starting to “learn”
  - Find Usages



# Why Java on the JVM?

- The tools
  - People have mixed reactions to refactoring tools
  - IDEs starting to “learn”
  - Find Usages
- The ecosystem



# Why Java on the JVM?

- The tools
  - People have mixed reactions to refactoring tools
  - IDEs starting to “learn”
  - Find Usages
- The ecosystem
  - One of if not the largest collections of FOSS libraries in existence



# Why Java on the JVM?

- The tools
  - People have mixed reactions to refactoring tools
  - IDEs starting to “learn”
  - Find Usages
- The ecosystem
  - One of if not the largest collections of FOSS libraries in existence
  - No language impedance





# Dat Tool





# Dat Tool



# Dat Tool

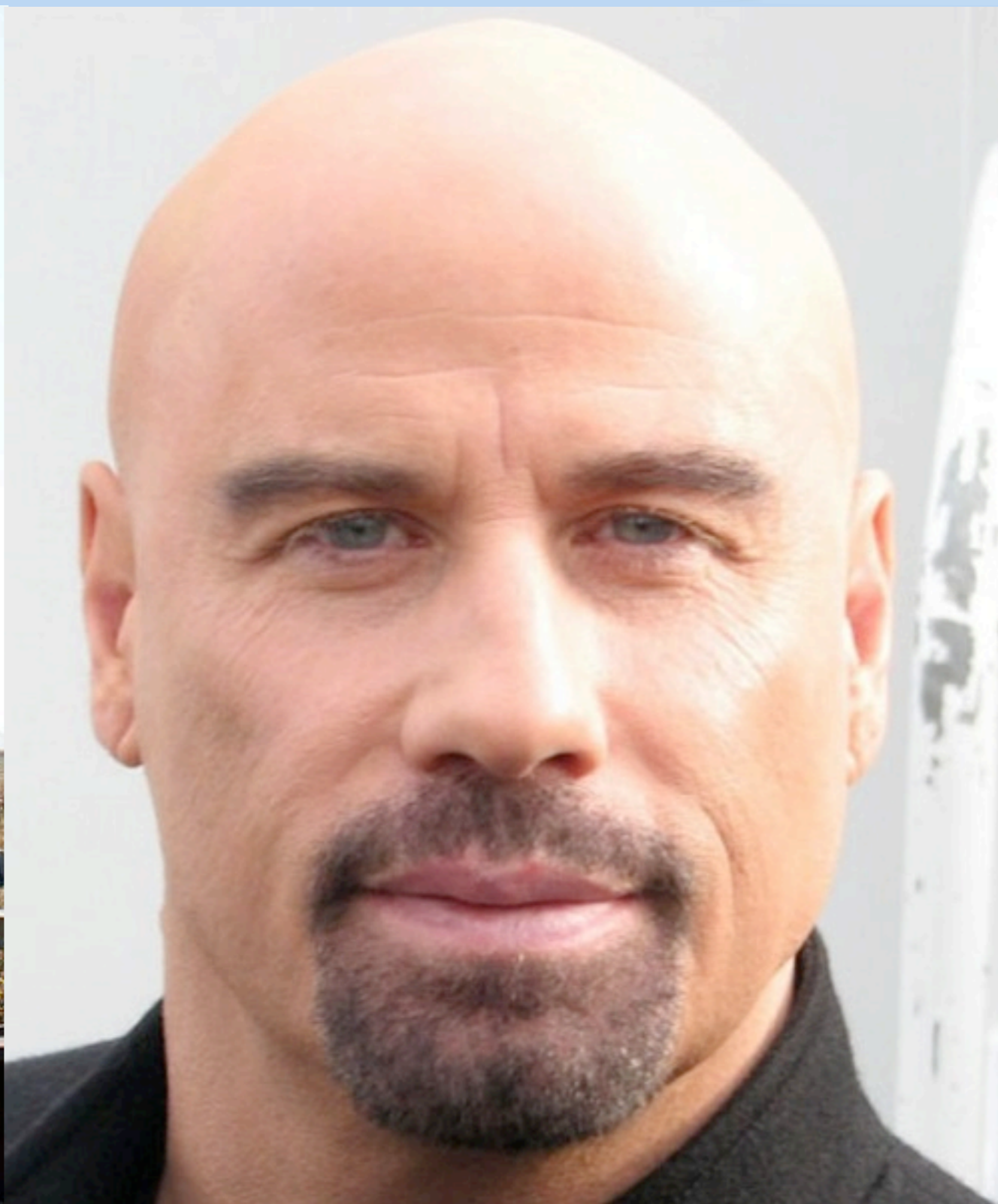


# Dat Tool





# Dat Tool



# Dat Tool



# Dat Tool



# Dat Tool

- Our job as developers is to implement something of business value



# Dat Tool

- Our job as developers is to implement something of business value
- Things that make us more productive are good





# Dat Tool

- Our job as developers is to implement something of business value
- Things that make us more productive are good
- It's OK to use tools to help you build things if they improve your productivity **and** further the roles of a developer



# Dat Tool

- Our job as developers is to implement something of business value
- Things that make us more productive are good
- It's OK to use tools to help you build things if they improve your productivity **and** further the roles of a developer
- Most IDEs are completely misunderstood by someone who has never tried them - *"I would miss my modal editing"*



Come at me Troll!



# Come at me Troll!

**But Java doesn't have closures!**



# Come at me Troll!

## **But Java doesn't have closures!**

- Java does have closures (imperfect closures to be exact)



# Come at me Troll!

## **But Java doesn't have closures!**

- Java does have closures (imperfect closures to be exact)
- Java does not have lambda expressions (yet)



# Come at me Troll!

## **But Java doesn't have closures!**

- Java does have closures (imperfect closures to be exact)
- Java does not have lambda expressions (yet)
- They're nowhere near as painful as people claim



# Come at me Troll!

## But Java doesn't have closures!

- Java does have closures (imperfect closures to be exact)
- Java does not have lambda expressions (yet)
- They're nowhere near as painful as people claim

```
final Channel channel = this.stateMachine.getApidChannel(realApid);
if (channel != null) {
    if (this.serviceManager.getAirDockService().registerDevice(realApid)) {
        channel.getCloseFuture().addListener(new ChannelFutureListener() {
            @Override
            public void operationComplete(ChannelFuture future) throws Exception {
                serviceManager.getAirDockService().removeDevice(realApid);
            }
        });
    }
}
```





Come at me Troll!



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

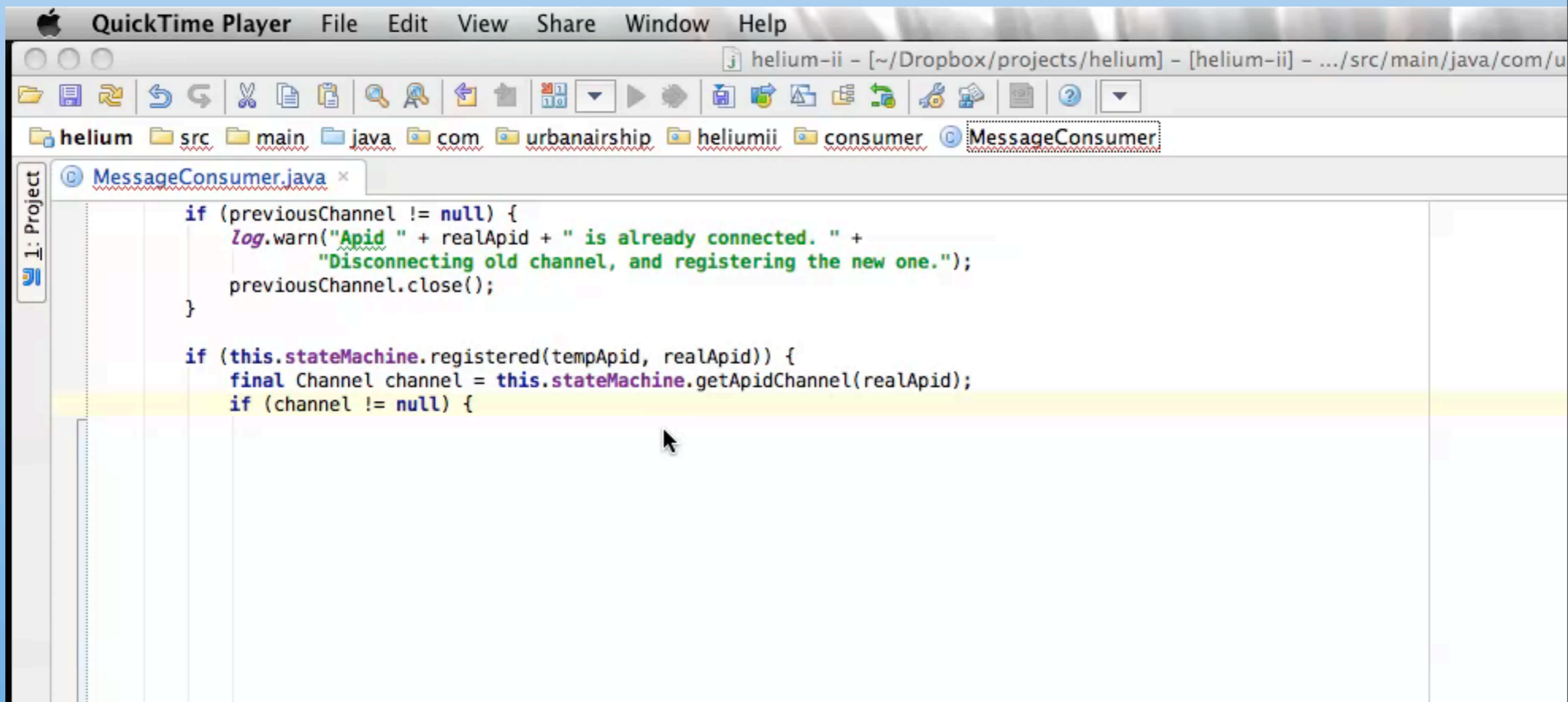
- Stop pounding the nail with your head



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- Stop pounding the nail with your head



The screenshot shows a code editor window titled "MessageConsumer.java" with the following Java code:

```
if (previousChannel != null) {
    log.warn("Apid " + realApid + " is already connected. " +
        "Disconnecting old channel, and registering the new one.");
    previousChannel.close();
}

if (this.stateMachine.registered(tempApid, realApid)) {
    final Channel channel = this.stateMachine.getApidChannel(realApid);
    if (channel != null) {
```

Come at me Troll!



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- `val numbers = Array<Integer>(1,2,3,4)`



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- `val numbers = Array<Integer>(1,2,3,4)`
- `val numbers = Array(1,2,3,4)`





# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- `val numbers = Array<Integer>(1,2,3,4)`
- `val numbers = Array(1,2,3,4)`
- `int[] numbers = new int[]{1,2,3,4};`



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- `val numbers = Array<Integer>(1,2,3,4)`
- `val numbers = Array(1,2,3,4)`
- `int[] numbers = new int[]{1,2,3,4};`

**Hey, that example is not fair!**



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- `val numbers = Array<Integer>(1,2,3,4)`
- `val numbers = Array(1,2,3,4)`
- `int[] numbers = new int[]{1,2,3,4};`

**Hey, that example is not fair!**

- `val str = "SPORTS!"`



# Come at me Troll!

**But Java makes me type so much and that hurts my delicate hands!**

- `val numbers = Array<Integer>(1,2,3,4)`
- `val numbers = Array(1,2,3,4)`
- `int[] numbers = new int[]{1,2,3,4};`

**Hey, that example is not fair!**

- `val str = "SPORTS!"`
- `final String str = "SPORTS!";`



# Come at me Troll!

**But Java has soooo much line noise it's hard to read!**

- Stop pounding the nail with your head
- The human brain simply doesn't work like that

i cdnuolt blveiee taht I cluod aulacilty uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuan mnid, aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it dseno't mtaetr in waht oerdr the ltteres in a wrod are, the olny iproamtnt tihng is taht the frsit and lsat ltteer be in the rghi t pclae. The rset can be a taotl mses and you can sitll raed it whotuit a pboerlm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Azanmig huh? yaeh and I awlyas tghuhot slpeling was ipmorantt! if you can raed tihs forwrad it.



# What We Can Change



# What We Can Change

- Java gets a bad rap, often for baggage in the ecosystem as a whole rather than the language specifically



# What We Can Change

- Java gets a bad rap, often for baggage in the ecosystem as a whole rather than the language specifically
- Some of this is due to big vendors in the space





# What We Can Change

- Java gets a bad rap, often for baggage in the ecosystem as a whole rather than the language specifically
- Some of this is due to big vendors in the space
- Some of it is because of the bloat



# What We Can Change

- Java gets a bad rap, often for baggage in the ecosystem as a whole rather than the language specifically
- Some of this is due to big vendors in the space
- Some of it is because of the bloat
- But usually, it's because we Java developers learn too slowly from things going on around us and get stuck in our ways



# What We Can Change



# What We Can Change

- **Do not** - view the world as a pattern waiting to happen



# What We Can Change

- **Do not** - view the world as a pattern waiting to happen
  - Patterns, Enterprise Patterns, Anti-patterns and please yes, buy my book



# What We Can Change

- **Do not** - view the world as a pattern waiting to happen
  - Patterns, Enterprise Patterns, Anti-patterns and please yes, buy my book
- **Do** - get your job done



# What We Can Change

- **Do not** - view the world as a pattern waiting to happen
  - Patterns, Enterprise Patterns, Anti-patterns and please yes, buy my book
- **Do** - get your job done
  - Your business sponsors don't care that you used `FlyweightSingletonFactoryDelegateVisitor.java`



# What We Can Change

- **Do not** - view the world as a pattern waiting to happen
  - Patterns, Enterprise Patterns, Anti-patterns and please yes, buy my book
- **Do** - get your job done
  - Your business sponsors don't care that you used `FlyweightSingletonFactoryDelegateVisitor.java`
  - Of course, some usage is fine but focus on writing code - patterns don't define correctness





# What We Can Change



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly
  - Be tempted into talk of AoP and cross-cutting concerns - your job isn't just coding, it's also sustaining and troubleshooting



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly
  - Be tempted into talk of AoP and cross-cutting concerns - your job isn't just coding, it's also sustaining and troubleshooting
  - Concurrency doesn't align w/ business concerns anyway



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly
  - Be tempted into talk of AoP and cross-cutting concerns - your job isn't just coding, it's also sustaining and troubleshooting
  - Concurrency doesn't align w/ business concerns anyway
  - Business domain doesn't have "logs" but OPS needs to



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly
  - Be tempted into talk of AoP and cross-cutting concerns - your job isn't just coding, it's also sustaining and troubleshooting
  - Concurrency doesn't align w/ business concerns anyway
  - Business domain doesn't have "logs" but OPS needs to
- **Do** - leverage proven, simple libraries to help



# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly
  - Be tempted into talk of AoP and cross-cutting concerns - your job isn't just coding, it's also sustaining and troubleshooting
  - Concurrency doesn't align w/ business concerns anyway
  - Business domain doesn't have "logs" but OPS needs to
- **Do** - leverage proven, simple libraries to help
  - Coda Hale's metrics library out of Yammer is essential





# What We Can Change

- **Do not** - be afraid to have non-domain code in your code
  - If latency to a system matters, measure that explicitly and overtly
  - Be tempted into talk of AoP and cross-cutting concerns - your job isn't just coding, it's also sustaining and troubleshooting
  - Concurrency doesn't align w/ business concerns anyway
  - Business domain doesn't have "logs" but OPS needs to
- **Do** - leverage proven, simple libraries to help
  - Coda Hale's metrics library out of Yammer is essential
  - Log4J, SLF4J, lots of good choices



# What We Can Change



# What We Can Change

- **Do** - learn from Erlang's immutable data structures



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe
    - Easy to put down the wire (Jackson, PBs) or into a schema-less store



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe
    - Easy to put down the wire (Jackson, PBs) or into a schema-less store
    - Just properties, no getFoo



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe
    - Easy to put down the wire (Jackson, PBs) or into a schema-less store
    - Just properties, no getFoo
    - No behaviors





# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe
    - Easy to put down the wire (Jackson, PBs) or into a schema-less store
    - Just properties, no getFoo
    - No behaviors
- **Do not** - assume your classes need to model "real world" things



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe
    - Easy to put down the wire (Jackson, PBs) or into a schema-less store
    - Just properties, no getFoo
    - No behaviors
- **Do not** - assume your classes need to model "real world" things
  - A class doesn't exist in the "real world", don't try and make it look like that



# What We Can Change

- **Do** - learn from Erlang's immutable data structures
  - Move data using immutable models
    - Fast, efficient, easy to reason about, thread safe
    - Easy to put down the wire (Jackson, PBs) or into a schema-less store
    - Just properties, no getFoo
    - No behaviors
- **Do not** - assume your classes need to model "real world" things
  - A class doesn't exist in the "real world", don't try and make it look like that
  - Bob doesn't have a save() method



# What We Can Change



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
- Leave no ambiguity about what impact an operation has on a back-end data store



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
  - Leave no ambiguity about what impact an operation has on a back-end data store
  - Measure latency to external services



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
  - Leave no ambiguity about what impact an operation has on a back-end data store
  - Measure latency to external services
  - Easier to reason about concurrency (implementation and consumption)





# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
  - Leave no ambiguity about what impact an operation has on a back-end data store
  - Measure latency to external services
  - Easier to reason about concurrency (implementation and consumption)
  - Go's behavior separate from data



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
  - Leave no ambiguity about what impact an operation has on a back-end data store
  - Measure latency to external services
  - Easier to reason about concurrency (implementation and consumption)
  - Go's behavior separate from data
- **Do not** - use magic, code generating systems that hide transactions across APIs



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
  - Leave no ambiguity about what impact an operation has on a back-end data store
  - Measure latency to external services
  - Easier to reason about concurrency (implementation and consumption)
  - Go's behavior separate from data
- **Do not** - use magic, code generating systems that hide transactions across APIs
  - Leaky abstraction - you call a method, a TXN happens



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs
  - Leave no ambiguity about what impact an operation has on a back-end data store
  - Measure latency to external services
  - Easier to reason about concurrency (implementation and consumption)
  - Go's behavior separate from data
- **Do not** - use magic, code generating systems that hide transactions across APIs
  - Leaky abstraction - you call a method, a TXN happens
  - When performance suffers, you will want to know what is going on but can't



# What We Can Change



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
- Learn from Go's keeping operations separate from data



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
  - Learn from Go's keeping operations separate from data
  - Return clones of data but nothing that has ties back in to a data store





# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
  - Learn from Go's keeping operations separate from data
  - Return clones of data but nothing that has ties back in to a data store
  - Easier to reason about, no ambiguity, pass as messages



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
  - Learn from Go's keeping operations separate from data
  - Return clones of data but nothing that has ties back in to a data store
  - Easier to reason about, no ambiguity, pass as messages
- **Do not** - use magic, code generating systems that hide transactions across APIs



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
  - Learn from Go's keeping operations separate from data
  - Return clones of data but nothing that has ties back in to a data store
  - Easier to reason about, no ambiguity, pass as messages
- **Do not** - use magic, code generating systems that hide transactions across APIs
  - A class doesn't exist in the "real world", don't try and make it look like the real world



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
  - Learn from Go's keeping operations separate from data
  - Return clones of data but nothing that has ties back in to a data store
  - Easier to reason about, no ambiguity, pass as messages
- **Do not** - use magic, code generating systems that hide transactions across APIs
  - A class doesn't exist in the "real world", don't try and make it look like the real world
  - Bob doesn't have a `save()` method



# What We Can Change

- **Do** - create coarse grained, transaction-aligned APIs operating on immutable models
  - Learn from Go's keeping operations separate from data
  - Return clones of data but nothing that has ties back in to a data store
  - Easier to reason about, no ambiguity, pass as messages
- **Do not** - use magic, code generating systems that hide transactions across APIs
  - A class doesn't exist in the "real world", don't try and make it look like the real world
  - Bob doesn't have a `save()` method
  - Nest models in behaviors, in models in behaviors



# What We Can Change



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do





# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do
  - Mapping it to some data isn't all that hard either



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do
  - Mapping it to some data isn't all that hard either
  - Test it and move on



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do
  - Mapping it to some data isn't all that hard either
  - Test it and move on
- **Do** - question configuration complexity and network abstractions



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do
  - Mapping it to some data isn't all that hard either
  - Test it and move on
- **Do** - question configuration complexity and network abstractions
  - If you go to the wire, that should be in your face



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do
  - Mapping it to some data isn't all that hard either
  - Test it and move on
- **Do** - question configuration complexity and network abstractions
  - If you go to the wire, that should be in your face
  - XML configurations are an immediate indicator of impending complexity



# What We Can Change

- **Do not** - be afraid to write SQL or actually new an object
  - Executing an SQL query is about the easiest thing you can do
  - Mapping it to some data isn't all that hard either
  - Test it and move on
- **Do** - question configuration complexity and network abstractions
  - If you go to the wire, that should be in your face
  - XML configurations are an immediate indicator of impending complexity
  - Realize IoC is not in and of itself a bad thing but you don't need a framework to accomplish it



# What We Can Change



# What We Can Change

- **Do not** - use XML for anything. Ever.





# What We Can Change

- **Do not** - use XML for anything. Ever.
- XML is pretty much good at nothing other than cementing unnecessary complexity at every level



# What We Can Change

- **Do not** - use XML for anything. Ever.
  - XML is pretty much good at nothing other than cementing unnecessary complexity at every level
  - XML is not code!



# What We Can Change

- **Do not** - use XML for anything. Ever.
  - XML is pretty much good at nothing other than cementing unnecessary complexity at every level
  - XML is not code!
- **Do** - use simple, standard mechanisms



# What We Can Change

- **Do not** - use XML for anything. Ever.
  - XML is pretty much good at nothing other than cementing unnecessary complexity at every level
  - XML is not code!
- **Do** - use simple, standard mechanisms
  - For configuration files, use properties files



# What We Can Change

- **Do not** - use XML for anything. Ever.
  - XML is pretty much good at nothing other than cementing unnecessary complexity at every level
  - XML is not code!
- **Do** - use simple, standard mechanisms
  - For configuration files, use properties files
    - Easier to use in modern automation environments like Puppet



# What We Can Change

- **Do not** - use XML for anything. Ever.
  - XML is pretty much good at nothing other than cementing unnecessary complexity at every level
  - XML is not code!
- **Do** - use simple, standard mechanisms
  - For configuration files, use properties files
    - Easier to use in modern automation environments like Puppet
    - Commons Config rocks



# What We Can Change

- **Do not** - use XML for anything. Ever.
  - XML is pretty much good at nothing other than cementing unnecessary complexity at every level
  - XML is not code!
- **Do** - use simple, standard mechanisms
  - For configuration files, use properties files
    - Easier to use in modern automation environments like Puppet
    - Commons Config rocks
  - For wire data, use PBs or JSON



# What We Can Change





# What We Can Change

- **Do not** - accept the “Enterprise” mentality



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process
    - Generally prohibit important things



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process
    - Generally prohibit important things
    - Not needed for scale



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process
    - Generally prohibit important things
    - Not needed for scale
- **Do** - write small, discrete, stand-alone services



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process
    - Generally prohibit important things
    - Not needed for scale
- **Do** - write small, discrete, stand-alone services
  - Easier to operate



# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process
    - Generally prohibit important things
    - Not needed for scale
- **Do** - write small, discrete, stand-alone services
  - Easier to operate
  - Easier to reason about





# What We Can Change

- **Do not** - accept the “Enterprise” mentality
  - Just say no to EARs, EJBs, and mostly WARs
    - More XML, complex build process, complex deploy process
    - Generally prohibit important things
    - Not needed for scale
- **Do** - write small, discrete, stand-alone services
  - Easier to operate
  - Easier to reason about
  - Strive for consistent approaches to all Java services



# What We Can Change



# What We Can Change

- **Do** - stop making web development so complicated



# What We Can Change

- **Do** - stop making web development so complicated
  - Conventional approaches are too difficult, especially for APIs



# What We Can Change

- **Do** - stop making web development so complicated
  - Conventional approaches are too difficult, especially for APIs
  - Consider Play or simply just embedding Jetty



# What We Can Change

- **Do** - stop making web development so complicated
  - Conventional approaches are too difficult, especially for APIs
  - Consider Play or simply just embedding Jetty
  - Embedding Jetty takes 16 lines of real code, add Jersey Annotations for fast, strongly typed HTTP endpoints and profit!



# What We Can Change

- **Do** - stop making web development so complicated
  - Conventional approaches are too difficult, especially for APIs
  - Consider Play or simply just embedding Jetty
  - Embedding Jetty takes 16 lines of real code, add Jersey Annotations for fast, strongly typed HTTP endpoints and profit!
- **Do not** - assume you need a WAR in a container to deliver or scale



# What We Can Change

- **Do** - stop making web development so complicated
  - Conventional approaches are too difficult, especially for APIs
  - Consider Play or simply just embedding Jetty
  - Embedding Jetty takes 16 lines of real code, add Jersey Annotations for fast, strongly typed HTTP endpoints and profit!
- **Do not** - assume you need a WAR in a container to deliver or scale
  - Sites of massive scale succeed without these mechanisms





# What We Can Change

- **Do** - stop making web development so complicated
  - Conventional approaches are too difficult, especially for APIs
  - Consider Play or simply just embedding Jetty
  - Embedding Jetty takes 16 lines of real code, add Jersey Annotations for fast, strongly typed HTTP endpoints and profit!
- **Do not** - assume you need a WAR in a container to deliver or scale
  - Sites of massive scale succeed without these mechanisms
  - Big vendors push a false sense of security



# What We Can Change



# What We Can Change

- **Do** - abuse **final**



# What We Can Change

- **Do** - abuse **final**
  - Simplifies closures, predicates, etc.



# What We Can Change

- **Do** - abuse **final**
  - Simplifies closures, predicates, etc.
  - Avoid public APIs relying on subclassing



# What We Can Change

- **Do** - abuse **final**
  - Simplifies closures, predicates, etc.
  - Avoid public APIs relying on subclassing
  - Clear expression of intent - you shall not change!



# What We Can Change

- **Do** - abuse **final**
  - Simplifies closures, predicates, etc.
  - Avoid public APIs relying on subclassing
  - Clear expression of intent - you shall not change!
- **Do not** - do it for performance reasons



# What We Can Change

- **Do** - abuse **final**
  - Simplifies closures, predicates, etc.
  - Avoid public APIs relying on subclassing
  - Clear expression of intent - you shall not change!
- **Do not** - do it for performance reasons
  - Your code may perform better but that's not the point





# What We Can Change

- **Do** - abuse **final**
  - Simplifies closures, predicates, etc.
  - Avoid public APIs relying on subclassing
  - Clear expression of intent - you shall not change!
- **Do not** - do it for performance reasons
  - Your code may perform better but that's not the point
  - Fall victim to the maybe extend effect



# What We Can Change



# What We Can Change

- **Do not** - use checked exceptions



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception
  - Terrible APIs



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception
  - Terrible APIs
- **Do** - have clean, expressive return types that indicate when something can go wrong that a consumer cares about



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception
  - Terrible APIs
- **Do** - have clean, expressive return types that indicate when something can go wrong that a consumer cares about
  - Make exceptions truly Exceptional - no catch blocks





# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception
  - Terrible APIs
- **Do** - have clean, expressive return types that indicate when something can go wrong that a consumer cares about
  - Make exceptions truly Exceptional - no catch blocks
  - Document what can go wrong in APIs



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception
  - Terrible APIs
- **Do** - have clean, expressive return types that indicate when something can go wrong that a consumer cares about
  - Make exceptions truly Exceptional - no catch blocks
  - Document what can go wrong in APIs
  - Return tuples (GO's value + error)



# What We Can Change

- **Do not** - use checked exceptions
  - No single non-Java language on the JVM honors them
  - Too prone to flow control by exception
  - Terrible APIs
- **Do** - have clean, expressive return types that indicate when something can go wrong that a consumer cares about
  - Make exceptions truly Exceptional - no catch blocks
  - Document what can go wrong in APIs
  - Return tuples (GO's value + error)
  - Google language design and C++ standards



# What We Can Change



# What We Can Change

- **Do** - monitor every single thing you may find interesting



# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles



# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles
  - Log files still matter - metrics and statistics need context



# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles
  - Log files still matter - metrics and statistics need context
- **Do not** - monitor JMX directly





# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles
  - Log files still matter - metrics and statistics need context
- **Do not** - monitor JMX directly
  - Most FOSS platforms are terrible at this, Most commercial ones too



# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles
  - Log files still matter - metrics and statistics need context
- **Do not** - monitor JMX directly
  - Most FOSS platforms are terrible at this, Most commercial ones too
  - JConsole is an awesome tool but not a monitoring or alerting platform



# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles
  - Log files still matter - metrics and statistics need context
- **Do not** - monitor JMX directly
  - Most FOSS platforms are terrible at this, Most commercial ones too
  - JConsole is an awesome tool but not a monitoring or alerting platform
- **Do not** - Assume “I can just hook up a profiler later”



# What We Can Change

- **Do** - monitor every single thing you may find interesting
  - Metrics and statistics are critical - 50th, 90th, 99th percentiles
  - Log files still matter - metrics and statistics need context
- **Do not** - monitor JMX directly
  - Most FOSS platforms are terrible at this, Most commercial ones too
  - JConsole is an awesome tool but not a monitoring or alerting platform
- **Do not** - Assume “I can just hook up a profiler later”
- **Do not** - Worry about performance of these things



# What We Can't Change

- No first class functions
  - Lambda expressions
- Long GC Pauses
- Bad LCD choices
  - Two reflections operations on private fields to get an FD? Really?
  - Spawning a process is painful compared to Python
- Type erasure



# Inspirations

- Stephan Schmidt: <http://codemonkeyism.com/generation-java-programming-style/>
- [https://github.com/technomancy/clojure-http-client/blob/master/src/clojure\\_http/client.clj](https://github.com/technomancy/clojure-http-client/blob/master/src/clojure_http/client.clj)
- My Python Co-workers





# Thanks!

- Urban Airship: <http://urbanairship.com/>
- We're hiring! <http://urbanairship.com/company/jobs/>
- Me @eonnen or erik at 