Execution in the Kingdom of Nouns

March 2006

by Steve Yegge

"They've a temper, some of them—particularly verbs: they're the proudest—adjectives you can do anything with, but not verbs—however, I can manage the whole lot of them! Impenetrability! That's what I say!" — Humpty Dumpty

Hello, world! Today we're going to hear the story of Evil King Java and his quest for worldwide verb stamp-outage[1].

***Caution**: This story does not have a happy ending. It is neither a story for the faint of heart nor for the critical of mouth. If you're easily offended, or prone to being a disagreeable knave in blog comments, please stop reading now.*

Before we begin the story, let's get some conceptual gunk out of the way.

The Garbage Overfloweth

All Java people love "use cases", so let's begin with a use case: namely, taking out the garbage. As in, "Johnny, take out that garbage! It's overflowing!"

If you're a normal, everyday, garden-variety, English-speaking person, and you're asked to describe the act of taking out the garbage, you probably think about it roughly along these lines:

```
get the garbage bag from under the sink
carry it out to the garage
dump it in the garbage can
walk back inside
wash your hands
plop back down on the couch
resume playing your video game (or what-
       ever you were doing)
```

Even if you don't think in English, you still probably still thought of a similar set of actions, except in your favorite language. Regardless of the language you chose, or the exact steps you took, taking out the garbage is a series of actions that terminates in the garbage being outside, and you being back inside, because of the *actions* you took.

Our thoughts are filled with brave, fierce, passionate actions: we live, we breathe, we walk, we talk, we laugh, we cry, we hope, we fear, we eat, we drink, we stop, we go, we take out the garbage. Above all else, we are free to *do* and to *act*. If we were all just rocks sitting in the sun, life might still be OK, but

[1] Beginning with the verb "to stamp out", which is being replaced by a call to `VerbEliminatorFactory.createVerbEliminator(currentContext).operate()`. But that's getting waaaaay ahead of ourselves...

we wouldn't be free. Our freedom comes precisely from our ability to *do* things.

Of course our thoughts are also filled with nouns. We eat nouns, and buy nouns from the store, and we sit on nouns, and sleep on them. Nouns can fall on your head, creating a big noun on your noun. Nouns are *things*, and where would we be without things? But they're *just* things, that's all: the means to an end, or the ends themselves, or precious possessions, or names for the objects we observe around around us. There's a building. Here's a rock. Any child can point out the nouns. It's the *changes* happening to those nouns that make them interesting.

Change requires action. Action is what gives life its spice. Action even gives spices their spice! After all, they're not spicy until you *eat* them. Nouns may be everywhere, but life's constant change, and constant interest, is all in the verbs.

And of course in addition to verbs and nouns, we also have our adjectives, our prepositions, our pronouns, our articles, the inevitable conjunctions, the yummy expletives, and all the other lovely parts of speech that let us think and say interesting things. I think we can all agree that the parts of speech each play a role, and *all* of them are important. It would be a shame to lose any of them.

Wouldn't it be strange if we suddenly decided that we could no longer use verbs?

Let me tell you a story about a place that did exactly that...

The Kingdom of Nouns

In the Kingdom of Javaland, where King Java rules with a silicon fist, people aren't allowed to think the way you and I do. In Javaland, you see, nouns are *very* important, by order of the King himself. Nouns are the most important citizens in the Kingdom. They parade around looking distinguished in their showy finery, which is provided by the Adjectives, who are quite relieved at their lot in life. The Adjectives are nowhere near as high-class as the Nouns, but they consider themselves *quite* lucky that they weren't born Verbs.

Because the Verb citizens in this Kingdom have it very, very bad.

In Javaland, by King Java's royal decree, Verbs are *owned* by Nouns. But they're not mere pets; no, Verbs in Javaland perform all the chores and manual labor in the entire kingdom. They are, in effect, the kingdom's slaves, or at very least the serfs and indentured servants. The residents of Javaland are quite content with this situation, and are indeed scarcely aware that things could be any different.

Verbs in Javaland are responsible for all the work, but as they are held in contempt by all, no Verb is ever permitted to wander about freely. If a Verb is to be seen in public at all, it must be escorted at all times by a Noun.

Of course “escort”, being a Verb itself, is hardly allowed to run around naked; one must procure a VerbEscorter to facilitate the escorting. But what about “procure” and “facilitate?” As it happens, Facilitators and Procurers are both rather important Nouns

whose job is is the chaperonement of the lowly Verbs “facilitate” and “procure”, via Facilitation and Procurement, respectively.

The King, consulting with the Sun God on the matter, has at times threatened to banish entirely *all* Verbs from the Kingdom of Java. If this should ever to come to pass, the inhabitants would surely need at least one Verb to do all the chores, and the King, who possesses a rather cruel sense of humor, has indicated that his choice would be most assuredly be "execute".

The Verb "execute", and its synonymous cousins "run", "start", "go", "justDoIt", "makeItSo", and the like, can perform the work of any other Verb by replacing it with an appropriate Executioner and a call to execute(). Need to wait? `Waiter.execute()`. Brush your teeth? `ToothBrusher(myTeeth).go()`. Take out the garbage? `TrashDisposalPlanExecutor.doIt()`. No Verb is safe; all can be replaced by a Noun on the run.

In the more patriotic corners of Javaland, the Nouns have entirely ousted the Verbs. It may appear to casual inspection that there are still Verbs here and there, tilling the fields and emptying the chamber pots. But if one looks more closely, the secret is soon revealed: Nouns can rename their execute() Verb after themselves without changing its character in the slightest. When you observe the FieldTiller till(), the ChamberPotEmptier empty(), or the RegistrationManager register(), what you're really seeing is one of the evil King's army of executioners, masked in the clothes of its owner Noun.

Verbs in Neighboring Kingdoms

In the neighboring programming-language kingdoms, taking out the trash is a straightforward affair, very similar to the way we described it in English up above. As is the case in Java, data objects are nouns, and functions are verbs[2]. But unlike in Javaland, citizens of other kingdoms may mix and match nouns and verbs however they please, in whatever way makes sense for conducting their business.

For instance, in the neighboring realms of C-land, JavaScript-land, Perl-land and Ruby-land, someone might model taking out the garbage as a series of actions — that is to say, verbs, or *functions*. Then if they apply the actions to the appropriate objects, in the appropriate order (*get* the trash, *carry* it outside, *dump* it in the can, etc.), the garbage-disposal task will complete successfully, with no superfluous escorts or chaperones required for any of the steps.

There's rarely any need in these kingdoms to create wrapper nouns to swaddle the verbs. They don't have GarbageDisposalStrategy nouns, nor GarbageDisposalDestinationLocator nouns for finding your way to the garage, nor PostGarbageActionCallback nouns for putting you back on your couch. They just write the verbs to operate on the nouns lying around, and then have a master verb, take_out_garbage(), that springs the subtasks to action in just the right order.

These neighboring kingdoms generally provide mechanisms for creating important nouns, when the need arises. If the diligent inventors in these kingdoms create an en-

[2] And variable names are proper nouns, attributes are adjectives, operators often serve as conjunctions, varargs are the pronoun "y'all", and so on. But this is all beside the point of our story.

tirely new, useful concept that didn't exist before, such as a house, or a cart, or a machine for tilling fields faster than a person can, then they can give the concept a Class, which provides it with a name, a description, some state, and operating instructions.

The difference is that when Verbs are allowed to exist independently, you don't need to invent new Noun concepts to hold them.

Javalanders look upon their neighbors with disdain; this is the way of things in the Kingdoms of Programming.

If You Dig a Hole Deep Enough...

On the other side of the world is a sparsely inhabited region in whose kingdoms Verbs are the citizens of eminence. These are the Functional Kingdoms, including Haskellia, Ocamlica, Schemeria, and several others. Their citizens rarely cross paths with the kingdoms near Javaland. Because there are few other kingdoms nearby, the Functional Kingdoms must look with disdain upon each other, and make mutual war when they have nothing better to do.

In the Functional Kingdoms, Nouns and Verbs are generally considered equal-caste citizens. However, the Nouns, being, well, nouns, mostly sit around doing nothing at all. They don't see much point in running or executing anything, because the Verbs are quite active and see to all that for them. There are no strange laws mandating the creation of helper Nouns to escort each Verb, so there are only exactly as many Nouns as there are Things in each kindgom.

As a result of all this, the Verbs have the run of the place, if you'll pardon the expression. As an outsider, you could easily form the impression that Verbs (i.e., the functions) are the most important citizens by far. That, incidentally, is why they're called the Functional Kingdoms and not the Thingy Kingdoms.

In the remotest regions, beyond the Functional Kingdoms, lies a fabled realm called Lambda the Ultimate. In this place it is said that there are no nouns at all, only verbs! There are "things" there, but all things are created from verbs, even the very integers for counting lambs, which are the most popular form of trading currency there, if the rumors speak truth. The number zero is simply lambda(), and 1 is lambda(lambda()), 2 is lambda(lambda(lambda())), and so on. Every single Thing in this legendary region, be it noun, verb or otherwise, is constructed from the primal verb "lambda"[3].

To be quite honest, most Javalanders are blissfully unaware of the existence of the other side of the world. Can you imagine their culture shock? They would find it so disorienting that they might have to invent some new nouns (such as "Xenophobia") to express their new feelings.

Are Javalanders Happy?

You might think daily life in Javaland would be at best a little strange, and at worst grossly inefficient. But you can tell how happy a society is through their nursery rhymes, and Javaland's are whimsically poetic. For instance, Javaland children oft recite the famous cautionary tale:

[3] The meaning of the verb "lambda" is allegedly "to lambda".

```
For the lack of a nail,
    throw new HorseshoeNailNotFoundException("no nails!");

For the lack of a horseshoe,
    EquestrianDoctor.getLocalInstance().getHorseDispatcher().shoot();

For the lack of a horse,
    RidersGuild.getRiderNotificationSubscriberList().getBroadcaster().run(
      new BroadcastMessage(StableFactory.getNullHorseInstance()));

For the lack of a rider,
    MessageDeliverySubsystem.getLogger().logDeliveryFailure(
      MessageFactory.getAbstractMessageInstance(
        new MessageMedium(MessageType.VERBAL),
        new MessageTransport(MessageTransportType.MOUNTED_RIDER),
        new MessageSessionDestination(BattleManager.getRoutingInfo(
                                        BattleLocation.NEAREST))),
      MessageFailureReasonCode.UNKNOWN_RIDER_FAILURE);

For the lack of a message,
    ((BattleNotificationSender)
      BattleResourceMediator.getMediatorInstance().getResource(
        BattleParticipant.PROXY_PARTICIPANT,
        BattleResource.BATTLE_NOTIFICATION_SENDER)).sendNotification(
          ((BattleNotificationBuilder)
            (BattleResourceMediator.getMediatorInstance().getResource(
            BattleOrganizer.getBattleParticipant(Battle.Participant.GOOD_GUYS),
            BattleResource.BATTLE_NOTIFICATION_BUILDER))).buildNotification(
              BattleOrganizer.getBattleState(BattleResult.BATTLE_LOST),
              BattleManager.getChainOfCommand().getCommandChainNotifier()));

For the lack of a battle,
    try {
        synchronized(BattleInformationRouterLock.getLockInstance()) {
          BattleInformationRouterLock.getLockInstance().wait();
        }
    } catch (InterruptedException ix) {
      if (BattleSessionManager.getBattleStatus(
           BattleResource.getLocalizedBattleResource(Locale.getDefault()),
           BattleContext.createContext(
             Kingdom.getMasterBattleCoordinatorInstance(
               new TweedleBeetlePuddlePaddleBattle()).populate(
                 RegionManager.getArmpitProvince(Armpit.LEFTMOST)))) ==
        BattleStatus.LOST) {
        if (LOGGER.isLoggable(Level.TOTALLY_SCREWED)) {
          LOGGER.logScrewage(BattleLogger.createBattleLogMessage(
            BattleStatusFormatter.format(BattleStatus.LOST_WAR,
                                         Locale.getDefault())));
        }
      }
    }
```

```
For the lack of a war,
    new ServiceExecutionJoinPoint(
      DistributedQueryAnalyzer.forwardQueryResult(
        NotificationSchemaManager.getAbstractSchemaMapper(
          new PublishSubscribeNotificationSchema()).getSchemaProxy().
            executePublishSubscribeQueryPlan(
              NotificationSchema.ALERT,
              new NotificationSchemaPriority(SchemaPriority.MAX_PRIORITY),
              new PublisherMessage(MessageFactory.getAbstractMessage(
                MessageType.WRITTEN,
                new MessageTransport(MessageTransportType.WOUNDED_SURVIVOR),
                new MessageSessionDestination(
                  DestinationManager.getNullDestinationForQueryPlan()))),
              DistributedWarMachine.getPartyRoleManager().getRegisteredParties(
                PartyRoleManager.PARTY_KING ||
                PartyRoleManager.PARTY_GENERAL ||
                PartyRoleManager.PARTY_AMBASSADOR)).getQueryResult(),
        PriorityMessageDispatcher.getPriorityDispatchInstance())).
      waitForService();

All for the lack of a horseshoe nail.
```

It remains wonderful advice, even to this very day.

Although the telling of the tale in Javaland differs in some ways from Ben Franklin's original, Javalanders feel their rendition has a distinct charm all its own.

The main charm is that the *architecture* is there for all to see. Architecture is held in exceptionally high esteem by King Java, because architecture consists entirely of nouns. As we know, nouns are *things*, and things are prized beyond all actions in the Kingdom of Java. Architecture is made of things you can see and touch, things that tower over you imposingly, things that emit a satisfying clunk when you whack them with a stick. King Java dearly loves clunking noises; he draws immense satisfaction from kicking the wheels when he's trying out a new horse-drawn coach. Whatever its flaws may be, the tale above does not want for *things*.

One of our first instincts as human beings is to find shelter from the elements; the stronger the shelter, the safer we feel. In Javaland, there are many strong things to make the citizens feel safe. They marvel at the massive architectural creations and think "this must be a strong design". This feeling is reinforced when they try to make any changes to the structure; the architectural strength then becomes daunting enough that they feel *nobody* could bring this structure down.

In addition to the benefits of a strong architecture, everything in Javaland is nicely organized: you'll find every noun in its proper place. And the stories all take a definite shape: object construction is the dominant type of expression, with a manager for each abstraction and a run() method for each manager. With a little experience at this kind of conceptual modeling, Java citizens

realize they can express *any* story in this style. There's a kind of "noun calculus" backing it that permits the expression of any abstraction, any computation you like. All one needs are sufficient nouns, constructors for those nouns, accessor methods for traversing the noun-graph, and the all-important execute() to carry out one's plans.

The residents of the Kingdom of Java aren't merely happy — they're bursting with pride!

StateManager.getConsiderationSetter ("Noun Oriented Thinking", State.HARMFUL).run()

Or, as it is said outside the Kingdom of Java, “Noun Oriented Thinking Considered Harmful”.

Object Oriented Programming puts the Nouns first and foremost. Why would you go to such lengths to put one part of speech on a pedestal? Why should one kind of concept take precedence over another? It's not as if OOP has suddenly made verbs less important in the way we actually think. It's a strangely skewed perspective. As my friend Jacob Gabrielson once put it, advocating Object-Oriented Programming is like advocating Pants-Oriented Clothing.

Java's static type system, like any other, has its share of problems. But the extreme emphasis on noun-oriented thought processes (and consequently, modeling processes) is more than a bit disturbing. Any type system will require you to re-shape your thoughts somewhat to fit the system, but eliminating standalone verbs seems a step beyond all rationale or reason.

C++ doesn't exhibit the problem, because C++, being a superset of C, allows you to define standalone functions. Moreover, C++ provides a distinct namespace abstraction; Java overloads the idea of a Class to represent namespaces, user-defined types, syntactic delegation mechanisms, some visibility and scoping mechanisms, and more besides.

Don't get me wrong; I'm not claiming C++ is “good”. But I do find myself appreciating the flexibility of its type system, at least compared with Java's. C++ suffers from problems causing reasonable-looking sentences to cause listeners to snap and try to kill you (i.e., unexpected segfaults and other pitfalls for the unwary), and it can be extremely difficult to find the exact incantation for expressing a particular thought in C++. But the range of succinctly expressible thoughts far exceeds Java's, because C++ gives you *verbs*, and who'd want to speak in a language that doesn't?

Classes are really the only modeling tool Java provides you. So whenever a new idea occurs to you, you have to sculpt it or wrap it or smash at it until it becomes a *thing*, even if it began life as an action, a process, or any other non-“thing” concept.

I've really come around to what Perl folks were telling me 8 or 9 years ago: “Dude, not everything is an object.”

It's odd, though, that Java[4] appears to be the only mainstream object-oriented language that exhibits radically noun-centric behav-

[4] And arguably C#, due to its similar roots.

ior. You'll almost never find an AbstractProxyMediator, a NotificationStrategyFactory, or any of their ilk in Python or Ruby. Why do you find them everywhere in Java? It's a sure bet that the difference is in the verbs. Python, Ruby, JavaScript, Perl, and of course all Functional languages allow you to declare and pass around functions as distinct entities without wrapping them in a class.

It's certainly easier to do this in dynamically typed languages; you just pass a reference to the function, obtained from its name, and it's up to the caller to invoke the function with the proper arguments and use its return value correctly.

But many statically-typed languages have first-class functions as well. This includes verbosely-typed languages like C and C++, and also type-inferring [functional] languages like Haskell and ML. The languages just need to provide a syntax for creating, passing and invoking function literals with an appropriate type signature.

There's no reason Java couldn't simply add first-class functions and finally enter the grown-up, non-skewed world that allows people to use verbs as part of their thought processes. In fact there's a JVM language called The Nice programming language that sports a very Java-like syntax, but also includes expressive facilities for using verbs: standalone functions, which Java forces you to wrap with Callbacks or Runnables or other anonymous interface implementation classes to be able to refer to them.

Sun wouldn't even have to break their convention of requiring all functions to be "owned" by classes. Every anonymous function could carry an implicit "this" pointer to the class in which it was defined; problem solved.

I don't know why Sun insists on keeping Java squarely planted in the Kingdom of Nouns. I doubt it's a matter of underestimating their constituency; they added generics, which are a far more complex concept, so they clearly no longer care deeply about keeping the language simple. And that's not a bad thing, necessarily, because Java's established now: it makes more sense to start giving Java programmers tools that let them program the way they think.

I sure hope they fix this, so I can take the trash out and get back to my video game. Or whatever I was doing.

noun like the word â€œfootâ€ in: â€œSteveâ€™s footâ€. (For the techy types: If b were a class object, it might calculate the square root of itâ€™s value every time itâ€™s assigned to (using â€˜operator=â€™ in C++) - and return a cached copy of that result in subsequent calls to sqrtâ€¦so â€œsqrt()â€ might easily be returning a property of b - not necessarily a process acting upon it - and.Â Yes, yes we do. The classic rant on this subject is Steve Yeggeâ€™s Execution in the Kingdom of Nouns , which talks about the limitations of a Java-style take on purist object-oriented programming. Programming languages exist to make computing understandable to human beings. Although most programming languages are not like human languages, we have a distinct tendency to try and use them as if they were. This gem is almost ten years old, but still so true: Execution in the Kingdom of Nouns, by Steve Yegge. â€œExecution in the Kingdom of Nounsâ€ is published by Dennis Traub. I'm still laughing over: 'The number zero is simply lambda(), and 1 is lambda(lambda()), 2 is lambda(lambda(lambda())), and so on. Every single Thing in this legendary region, be it noun, verb or otherwise, is constructed from the primal verb "lambda".' probably_wrong on July 3, 2014. Yeah, well, I wasn't precisely laughing when I saw it on my Formal Languages and Computability course... The Kingdom of Nouns - Free download as PDF File (.pdf), Text File (.txt) or read online for free. In the Kingdom of Javaland, where King Java rules with a silicon fist, people aren't allowed to think the way you and I do. In Javaland, you see, nouns are very important, by order of the King himself. Nouns are the most important citizens in the Kingdom.Â Nouns are the most important citizens in the Kingdom. They parade around looking distinguished in their showy finery, which is provided by the Adjectives, who are quite relieved at their lot in life. The Adjectives are nowhere near as high-class as the Nouns, but they consider themselves quite lucky that they weren't born Verbs.