**Cameron Purdy:**

ok, starting new thread, looping in Rob Lee

Mark: "I'd like to hear more about what you envision for Traits, I'm not picturing it yet..."

**Mark Falco:**

Oops, guess I responded on the wrong thread. Sorry/hi Rob :)

**Cameron Purdy:**

Trait is like Java "extends", except more like interfaces in its ability to be applied horizontally

**Rob Lee:**

I'm obviously missing context, but this sounds like aspects?

**Cameron Purdy:**

So, a Trait "extends" some specified type, but a type is defined by its properties and methods

So I could build a Trait called ChangeListener that "extends" a type called Property

Then declare a property "@ChangeListener int x;"

**Gene Gleyzer:**

If it's not parameterized, it will require an injection

Outside of trivial use cases

**Cameron Purdy:**

Or define a Trait called Loggable that extends type {to<String>()} i.e. Any type that can convert to a String, with method "log() {Sys.out.println(this);}"

You're thinking about stateless ones, aka "mixins"

**Gene Gleyzer:**

Yes

**Cameron Purdy:**

But yes injection is the replacement for "static" eg System.out

so, for sake of argument, any type T is defined as {{P1, P2, …, Pn} U {M1, M2, …, Mn}}

**Mark Falco:**

So any object having those is useable as that type?

**Cameron Purdy:**

so any type T2 is type T1 iff T2 ⊇ T1

so when I define a trait (or a mix-in) extending some type T, it just means that within that trait definition, the “this” will be of type T

**Gene Gleyzer:**

Assuming the property accessibility is equivalent

**Cameron Purdy:**

so yes, you can pass any type T2 transparently to anything declared to take type T1 iff T2 ⊇ T1

(i’m assuming unicode is working here ..)

**Mark Falco:**

Yes it is :)

**Cameron Purdy:**

accessibility is a bit different than in e.g. Java, though .. when one defines some things as being public / protected / private, what is actually occurring is that 3 different interfaces are being defined

so within that context, there are 3 different this’s

there is a this:public, a this:protected, and a this:private

if, for example, i have a method on type T called “public T giveMeTheKeysToTheKingdom()” and it simply does “return this:private;”, then you can imagine how easy it will be to call private members on T .. 😉

in other words, whatever is present in the vtabl that one has, those things are callable, and whatever is not present, those things “do not exist”

i.e. there is no “access control” per se

(the only sane security model: no security model)

i.e. design out the need for a runtime security system

**Mark Falco:**

Is there casting?

**Cameron Purdy:**

kind of …

the base object type has one property and 2 parameterized methods

**Mark Falco:**

Or only casting within the vtable you can see?

**Cameron Purdy:**

property is Class

methods are “T as<T>()” and “T to<T>()”

so yes, one can always cast within the vtabl that one has, unless that cast operation is explicitly overridden (which would seem silly)

i.e. a cast is the “as” method

long longvalue = intvalue.as<long>();

**Mark Falco:**

Ha

But cute :)

**Cameron Purdy:**

not sure how to translate the “Ha” portion

certainly different from java tho

**Mark Falco:**

So that gives you back the same object, so it is an int backed by the long

**Cameron Purdy:**

possibly. rules differ slightly for immutable values like int/long

however, the goal is that the cast could (in reality) be literally a no-op

i.e. something understood by the native compiler and completely optimized out

so for example, i can take “int n” and do something like: “return n & 1;”

or i could instead do “return n.as<bit[]>()[0];"

the bit array never actually exists as an object (assuming that it is not potentially escaped into any space that would treat it as anything other than a bit[])

it makes it very efficient to write decent code around what we consider primitive types, all while being able to treat them as real objects (which they are, except that the vtabl is hard-wired and the pointer to the vtabl is implicit and the reference to the object is actually its value)

lessons inspired by hotspot

(but not learned by hotspot .. yet)

ok, admit it already .. this is more fun than building Coherence .. or building the hardware version of Coherence with flash 😉

**Mark Falco:**

:)

**Rob Lee:**

:-)

**Cameron Purdy:**

serious packet delays from the west coast 😉

**Rob Lee:**

You know me, always trying to keep up...

**Cameron Purdy:**

on an unrelated note, “null” is not a subtype of all types like it is in Java. Instead, Nullable is a type, and “null” is an instance (perhaps \_the\_ instance) of Nullable. one cannot simply assign null to a reference, though, because the type of the reference is not Nullable unless specified as such. however, it is possible for a type to be defined as being one type or another, e.g. something like “T1 = T2 || T3;” … as such, a type can be “String || Nullable” which is a bit wordy, so instead the short-hand is “String?” … btw this syntax is now in C# as well

made a few good advances friday and this weekend. first “aha” moment was realizing that the Object interface has a property “public Type type;” not “public Class class;” if you will … the “type” is the interface as defined by methods and properties, not something that has a “name” and “package” etc.

i.e. it’s always possible to ask an object what its interface (type) is, and it’s always possible to cast it (.as<T>()) and it’s always possible to convert it (.to<T>()), but further than that, the object is (all other things equal) opaque

gene - this goes to the question of what one can explore given a reference, e.g. can they ask for the class / package / module etc.? (and from that, attempt to enumerate and navigate to other classes / packages / etc.) … the answer is apparently “no”.

**Rob Lee:**

Different between .as<T>() and .to<T>()?

Err difference

**Cameron Purdy:**

simple answer: in the case of mutable objects, .as<T>() can return a reference to the same object, which .to<T>() will not.

more complex answers available, but it comes down to the difference between a cast and a conversion.

**Gene Gleyzer:**

So Type doesn't have an identity?

**Cameron Purdy:**

obvious example of .to<T>() is .to<String>()

i am suggesting that Type does not have an identity separate from that defined by the contents of its interface, i.e. its identity is solely its interface

**Rob Lee:**

Type sounds like it is a function of all mixed traits/available methods etc, rather than a named programmatically entity

What is the behavior of to<T>() when the instance has no conversion? And how is the programmer to know?

**Cameron Purdy:**

reasonable question. obvious answer is that for some values of T, there will exist a method .to<T>(), which is quite apparently to a dev tool for compile time checking. less obvious is the ability to handle specific sub-classes of T, e.g. to handle .to<String>() in the .to<Object>() implementation … for that, I don’t (currently) have an answer of how an IDE would know … but runtime behavior of an unsupported type T being specified for .to or .as would seem to be a runtime exception

**Mark Falco:**

ClassToException :)

**Cameron Purdy:**

a second, seemingly unrelated decision that i came to is a new class category, called singleton. (existing categories include: class, something akin to java interface, trait, mixin, and enum)

(oops, i forgot “value”)

**Mark Falco:**

I’m curious if the to<T>() method is ultimately a good idea.  It feels a bit like Object.hashCode() which seemed like such a good idea, but also flawed in comparison to externalizing it with a Hasher<T>.

**Cameron Purdy:**

separate topic from all of the serious stuff: string literals.

i am generally trying to simplify the lexical structure of literals. strings themselves will look about the same. i am considering introducing a new “multi line string literal”, i.e text that spans multiple lines

**Rob Lee:**

Will make it harder for the source compiler to determine and communicate a relevant error location

**Cameron Purdy:**

the two use cases that i have identified for this are orthogonal. the first is that a significant amount of text needs to go into a string, but is doesn’t look good in the code having one line of the code exceed the 120 character right border limit by a few thousand characters. today, you’d solve that by ending each line with a quote mark and starting the next line with a plus and a quote mark to re-start the literal.

the second use case is for actual text, i.e. information in textual form that flows over multiple lines. in this case, the end of line in the source could very well be the end of line in the multi-line string literal

rob - no, the line number / offset is handled much earlier in the lexical processing.

The first stage of lexical analysis tracks the line number and the line offset in order to identify exact locations within the source code; both the line number and line offset are zero-based. Any location within the source can be identified by a combination of the line number and the line offset, while a selection of the source is identified by a starting location (inclusive) and an ending location (exclusive). The location tracking uses the definition of the LineTerminator sequence to determine when to increment the line number and reset the line offset:

LineTerminator:

U+000A

U+000B

U+000C

U+000D U+000Aopt

U+0085

U+2028

U+2029

Each RawCharacter in the character stream is evaluated to determine if it is the beginning of a LineTerminator sequence; if it is, then reading past the LineTerminator sequence causes the line number to be incremented and the line offset to be reset to zero. Reading any other RawCharacter (that is not the beginning of a LineTerminator sequence) causes the line offset to be incremented and does not change the line number. Note that in the case of the two-character LineTerminator sequence also known as “CR/LF”, the line number and line offset are undefined in between the contiguous characters U+000D and U+000A.

**Gene Gleyzer:**

And the quotes/plus are handled by an editor

**Cameron Purdy:**

well, the quotes and pluses are how it’s done today e.g. in java

i’m thinking of something that doesn’t need the quotes and pluses. i’ve mocked up a few possible examples, but i doubt that they’ll survive coming through iMessage

**Gene Gleyzer:**

And IntelliJ handled it seamlessly

**Rob Lee:**

...but emacs... :-)

**Mark Falco:**

can you email the examples

**Cameron Purdy:**

sure. but i was looking for some feedback on the concept itself. i think large amounts of text make for difficult code reading at times, but i am also aware how easy it is to make things like this unintentionally worse

what is emacs? is that a new apple computer?

ok, see gmail. mark - see gene’s 😉

i didn’t see any responses / comments from the stuff i sent yesterday, so i’ll have to pile on some more today. specifically, i’d like to introduce you to the current thinking on the ontology (or perhaps taxonomy) of this language. specifically, at the highest level, what is the conceptual model of the language, and what are the constructs that are used to support those concepts in the language?

**Mark Falco:**

feel like introducing it over lunch?

**Rob Lee:**

Don't think you guys can get here in time for lunch...

**Cameron Purdy:**

i want to start with the items that someone will actually be able to craft. not surprisingly, the most obvious one is called a “class”, which is the embodiment of a instantiable, realizable type.

yes, i can do lunch .. but not in california ;-)

**Mark Falco:**

how about burlington

just a bit closer

**Cameron Purdy:**

ok .. are you thinking at the office? or ..?

**Mark Falco:**

Musk needs to get that hyper loop on line so Rob can make it

we can easily do better then the office, any thoughts?

**Cameron Purdy:**

man, if anyone can do the impossible, that guy (Musk) seems to be at the head of the line each time

**Mark Falco:**

looking at my calendar I’ve got a call from 1-2. so either early or late would work of rme

**Cameron Purdy:**

early works

there’s a place in the Burlington Marketplace that does burgers and salads and stuff

**Mark Falco:**

five guys?

**Cameron Purdy:**

b.good

**Mark Falco:**

sure, that works

**Cameron Purdy:**

cool .. say 11:45 or so?

**Mark Falco:**

sounds good, see you then

**Gene Gleyzer:**

Sorry, have to skip this - on the Cape

**Cameron Purdy:**

gene - maybe you can have lunch with rob then

**Gene Gleyzer:**

But please feel free to assign any of those tasks to me :)

**Cameron Purdy:**

after a long couple of days working through some design inconsistencies combined with missing items from my wish list, i have come to a shocking conclusion: i am going to reintroduce the concept of “static”. before you burn me at the stake, hear me out. what i need in one case is an elegant way to describe a named constant, such as we would currently use “static final” for in java, for example. in another case, i need an ability to describe a named constant of type function (which, in my design thus far, is a method that is already bound to a “this”). here again, the similarity to java’s “static void foo()” is demonstrable. so what i am proposing is to use the term “static” so that developers coming from Java or C# will find it familiar, even though the underlying concepts (global scope vs. named constants) are seemingly different at the surface level, because i recognize that for the use case that i am suggesting, the java analogue is plenty good, and well understood by many developers already.

by borrowing the keyword “static”, i can easily avoid any special “function” keyword or similar, and avoid expressing it as a “state constant whose type is a function”, and instead just modify the “method signature” with the keyword “static”.

so the keyword would only be usable with a “property declaration” of a type that is “value” (i.e. immutable at the end of construction) or the aforementioned functions.

lastly, as “enum” is an extension of “value”, the concept holds for the named enum values as well.

lest i forget, the initial challenge that drove me into this swamp was the design of the “module” and “package” types. i believe that i have now been able to reconcile them both as extensions to the “singleton” type (keyword), which is itself an extension to the “class” type (keyword). how this is related to “static” is a long discussion in and of itself, but that is nowhere near as important as the conclusion that “module” and “static” can leverage all of the existing design building blocks instead of standing separate from them as exceptional cases.

and with that, i shall retire for the evening. (it’s a pennsylvania dutch saying .. it means “go to bed”)

actually, i should point out that “module”, “package”, and “function” were the last 3 structures that i still had to resolve (in terms of how they fit into the type system and appear in the syntactical system), so if this holds water, it means that i have “closed the loop” at least on one level …

writing from Cape Software 😊

quick recap of type system: developer builds a “module”, which contains “package” (recurse), which contains various forms of “class” (recurse). forms of class are (keywords): class, interface, value, singleton, enum (which is also a singleton and also a value), trait, mixin. two special forms of class: module and package (both of which are also singleton and value).

so a “type” can be any of those. each is represented by a constant in the module’s constant pool. every type is referenceable by name. however, when a name is used to reference a constant in the constant pool, that name may have a “default de-reference” associated with it. for example, if App.Util is a package, then a reference to the package can be obtained simply by using its name: “Package pkg = App.Util;” for example. But if App.Util.DefaultComparator is a singleton, then the name returns the instance: “Comparator cmp = App.Util.DefaultComparator;” (or even “Comparator cmp = DefaultComparator;” if it has been imported etc.). In the case of a singleton, to access the class, one must use the “!” post operator: “Comparator cmp = DefaultComparator!;”

Interestingly in the above example, App.Util is a package, but a package is a type, so “Package pkg = App.Util;” actually obtains the singleton instance of the App.Util class. To get the “class” of the Util “package” instead, it would be “Package pkg = App.Util!;”

(The same syntax is used with methods, properties, variables, etc. For example, if the Point class has a property “int x”, then in the code “point.x = 1;” the “point.x” is a reference to an instance of Property, but the “default dereference” is a call to the property’s set() method, i.e. as if the developer had written: “point.x!.set(1);”. similaly, “int value = point.x;” is the same as “int value = point.x!.get();”

another way to think about it is to imagine the “!” is a way to tell the compiler, “no dummy, that’s not what i meant .. get me the thing i named!”

FWIW - this closes out so many questions / topics that gene and i were going through. as in “it’s done; i’m happy.”

oops .. looks like i may have gotten traits and mixins backwards .. from wikipedia (which is never wrong) ”In contrast, mixins include full method definitions and may also carry state through member variable, while traits usually don't.”

holy cow .. <http://arstechnica.com/information-technology/2016/07/how-oracles-business-as-usual-is-threatening-to-kill-java/>

**Rob Lee:**

Yeah. I tweeted about this a couple wks ago :-(

**Gene Gleyzer:**

He's talking about jee, which is absolutely deserves to die

**Rob Lee:**

<https://www.change.org/p/larry-ellison-tell-oracle-to-move-forward-java-ee-as-a-critical-part-of-the-global-it-industry>

**Cameron Purdy:**

when java ee is allowed to die, a big pile of money goes to risk in the WLS business

the component specifications themselves haven’t moved forward; java ee as a big uber platform hasn’t been the bugbear it was for 10 years now

one question that comes up at this point is what one catches when one is expecting an exception. does one catch a Class or a Type ?

**Mark Falco:**

If an object can be both a class and type aren't both legal?

**Cameron Purdy:**

the issue is that you can’t ask an object what its class is, but you can ask what its type is

however, that is arguably irrelevant, since the catch clause isn’t asking, it’s something implemented on your behalf

**Mark Falco:**

Gotcha, so catching by class would apparently be limited to the thing the object is willing to expose itself as via as<T>

For what it's worth I think c++ disallows catching via type conversion

**Cameron Purdy:**

to answer your specific question, any combination of methods and properties represents a type, as does any combination of types. every class has a type. (more correctly, there are several types defined for every class, such as its public, protected, and private interfaces)

**Gene Gleyzer:**

Isn't it a question of "what type can one throw"?

**Cameron Purdy:**

catching by class is what java does, which generally i like. my concern with catching by type is that it would become far easier to accidentally catch things. for example, most exceptions in Java have the same “type” as Exception or RuntimeException ..

**Gene Gleyzer:**

Is there a common base for it like Throwable?

**Cameron Purdy:**

Rob - curious how you implemented exceptions when you built the JIT. did you use side tables for stack unwinding like C++ typically does, or did you simply expand on the function return values to avoid having to build a separate exception handling imply

impl not imply

gene - that is part of the riddle that i’m trying to figure out.

what’s the best name for this enum?

enum Ordered (LessThan, Equals, GreaterThan);

it’s technically an ordered field, is it not?

(Gene - Mr. Math?)

or perhaps: enum Ordered (Lesser, Equal, Greater);

the irony of trying to document … this one is from the Type property on the Object class: “The Type property provides the type of the reference that was used to obtain the value of the Type property.”

**Rob Lee:**

Long latency - the way we did exceptions in the Oracle JIT was we code generated the exception dispatch table and associated trampolines/unwinds. So a "throw" would be a call into a stub that does a lookup for the offset of the exception dispatcher region for the method, and a jump to that

**Cameron Purdy:**

glad to hear you remember how to write it. .. gonna need it again soon 😉

i have been struggling with some aspects of an area that gene and i have been working through. i think i am now just coming to terms with it.

gene - this goes back to the discussion of L values and R values that we were having yesterday, and how programming languages tend to hide various L- and R-value-specific behavior (such as automatic “load” aka de-referencing in one instance, and automatic store in another) behind whatever source code name represents the R- / L-value

one thing that i’ve been struggling with, though, is the connection between the two, i.e. what aspects are hiding in the reference itself. unlike typical single-dispatch languages (Java, C++ for the most part), the reference isn’t just a pointer. that means you can have differing references to the same value, to the same object, etc.

and that information is significant, because what the object chooses to expose of itself is encoded into that reference.

and how the mechanics of the reference actually work are also encoded there ..

btw … <http://ecstasylang.org/>

**Mark Falco:**

:)

**Cameron Purdy:**

oh, and also .. <http://xqiz.it/>

on the language design side, i’ve run into some giant obstacles over the past 48 hours, and i’ve slowly worked through most of them.

a big part of closing the loop is actually coming to terms with some of the realities of trade-offs that we’ve made thus far and the implications that came out of those (a few of which i was not expecting)

a couple hours of misek & co.

the “x” is supposed to be reminiscent of an app icon e.g. on a phone

**Rob Lee:**

Better

**Cameron Purdy:**

i’ve been working pretty much non-stop on resolving the issue that i encountered a few days ago. it was fairly existential. the long and the short of it is this: context matters.

one of the general rules that gene and i settled on early (indeed, one of the things that really sets this apart from other approaches) is that the reference includes the rights.

for lack of a better way to explain it, the reference for all practical purposes contains the vtabl itself.

so if you have access to a reference with lots of stuff in its vtabl (e.g. “this”), you can hand out that reference (or any part thereof) to whomever you want to, e.g. as a parameter, as state on some object, as a return value, etc.

this is particularly handy for addressing some of the most “corrupt” issues in languages such as C++ (“friends”) and Java (serialization)

but it also potentially opens a giant wormhole. inadvertently. here’s why:

if i “new” an object, the reference that i get to the new object is based on my locality to the thing i’m creating. for example, if i’m inside class X and i new an instance of class X, i naturally end up with a reference that includes access to all of the private members of X. if i’m just in the same package as X, I don’t get the private stuff, but i do get everything else including package level and module level stuff.

and so on. the net result is that an awful lot of the references floating around will have an awful lot of unintended access represented by those references.

as a rule of thumb, to answer the oft asked question of “whom are you trying to defend from”

the answer is fairly simple within a module: no one. in other words, within the same module, we don’t even “worry” about private vs. public or anything in between.

similarly, within a container (i.e. what the code thinks of as its entire universe), we are not trying to defend from the creator of the container.

it’s a bit of a grey area within a container but across module boundaries. we’re “in the same universe”, but at the same time, i can imagine that developers will be uncomfortable knowing that arbitrary modules could be violating their “privacy”. this may be a grey area primarily because of perception issues.

**Rob Lee:**

You need to protect across module boundaries though within a container

**Cameron Purdy:**

lastly, in terms of “what are you trying to defend from?”, the real answer is this:

**Rob Lee:**

If you want modularity isolation

Yes

**Cameron Purdy:**

we are ABSOLUTELY AND INSANELY trying to defend everything outside of a created container from the contents of that container.

and since only dependency injection introduces references into that container, it is incredibly important that those references cannot leak more information than the “god” outside of the container wishes to inject into it …

my solution is currently undergoing testing in my head. it will have some runtime cost.

my hope is that the cost will be nominal, but it will be an added indirection when a reference is first used within a frame.

to back up for a second, i was already planning to allow running code to ask where it was being invoked from, and in two different ways. the first is “who is calling this method?” and the second is “who is calling into this this?” i.e. the second allows a call to x.foo that then calls x.foo2 that calls x.foo3 to still know that x was called from y.

however, the answer isn’t “who is calling?”, but rather, “how far away is calling?”

with an enum something like {CLASS, PACKAGE, MODULE, FOREIGN, UNKNOWN} .. names still TBD, but FOREIGN being another module in the same container and UNKNOWN being God or delegate thereof

**Gene Gleyzer:**

Why is it a question that deserves an answer?

**Cameron Purdy:**

so to go back to the problem i was trying to solve, it is still possible for the reference to carry the vtabl information, i.e. exactly what it allows access to, but in addition to that, (i am suggesting that) it will be possible for the reference to instead carry the type information with an indicator that the access is relevant to who the caller is …

**Gene Gleyzer:**

In Java it's only asked by the securi

**Rob Lee:**

You want to allow to programmatically conditionality behavior based on caller "distance"?

**Cameron Purdy:**

Gene - good question. i have some ideas as to why, but in general, my answer to “why” thus far has been that i needed it myself to implement some specific things

so in the end it may not have to be a detail that is made available to joe programmer. on the other hand, my assumption thus far is that it will be … however, i’m not wed to the concept.

rob - exactly. if a reference works for me to access package private functionality because i’m in the same package, it wouldn’t allow someone in a different module with the same reference to access that same functionality.

i.e. there’s a de-ref that has to occur in order to resolve the reference within the frame.

so far, this is the only solution that i’ve been able to get to line up with the problem. i’ve been wracking my brain for days, running some sims in my head, and nothing else quite resolves the problem.

although in interest of full disclosure, i was tempted to significantly curtail the specificity of locale (drop everything except private/protected and public), and not worry about references carrying too much capability within a container boundary

i.e. i have made a fairly reasonable case in my head for allowing the wild west w.r.t. runtime accessibility within a container, but it basically means that all references would have to be far “wider” than their APIs would suggest (so when they are passed from areas of low locality to areas “close to” the private level, those references could be used to access module-private and package-private info, for example)

it’s really hard to explain all of this in a chat window … but trust me when i say it’s an “existential issue” for the design.

**Gene Gleyzer:**

I thought that unless explicitly specified, a reference loses all non-public methods any time it goes out of scope and that fidelity cannot be retrieved

**Cameron Purdy:**

that was another model i was playing in my head. it would certainly be handy if that were sufficient …

**Gene Gleyzer:**

Are you talking about what happens beyond that?

**Cameron Purdy:**

so what does “out of scope” mean?

**Gene Gleyzer:**

Out of module

Are you thinking about accessibility of an object instantiated by a foreign module?

**Cameron Purdy:**

of course

**Gene Gleyzer:**

And the issue is that when it comes back in scope we need to make it "richer", right?

**Cameron Purdy:**

that is my postulation

particularly because it may “come back in” with far less than even its public API, ie. from a “secure container”

(where each injected reference goes through a strict .as<T>() narrowing by declared injection type)

this problem should have been obvious to me a long time ago, but i didn’t have enough of a pretend code base running in my head to encounter it

the “solution” i’m proposing keeps everything quite square

but it does conceptually involve additional work at runtime (at least when a ref is used to access methods which are not public), which is annoying to me, because that is one of the things that i had hoped to be able to completely avoid (i.e. by “encoding the vtabl into the reference”)

anyhow, i will continue to focus on this issue until i have it perfectly solved. it’s fairly central .. not an area that i’m willing to leave bleeding

did you like the logo(s)?

i should have mentioned … the “x” portion would be re-used across [xqiz.it](http://xqiz.it) and [xtclang.org](http://xtclang.org)

giesi isn’t enjoying the fact that i’m working so much again … and when i’m not “working”, i’m still working in my head, and it’s driving her nuts.

it’s hard to turn it off though, when i’m working through a tough problem. it feels pretty good to be thinking again and creating stuff again .. now if i could only get this design completely nailed down and start cutting actual code again (not just in my head)

oh .. interesting article i’m working through right now that seems related to some topics gene and i were discussing: <http://tpolecat.github.io/2015/04/29/f-bounds.html>

am i the only one who has a tough time being turned on by scala?

so it turns out that the issue i was describing is solely a result of the introduction of additional levels of privacy (package and module level). when i remove them, the issue disappears entirely. (i think this is what gene was asking about.)

so my $.02 is that this comes down to a decision as to whether module and package privacy are important enough to introduce the additional complexity that i was describing last night, vs. whether private/protected and public are sufficient.

Any feedback?

**Gene Gleyzer:**

Yes; quite a bit. Not over text thou

**Cameron Purdy:**

Drop by?

**Rob Lee:**

A bit far 😕

**Mark Falco:**

Tonight doesn’t work for me, but Wednesday afternoon should

**Gene Gleyzer:**

Me too

**Mark Falco:**

Rob, that gives you enough time as well 😉

**Rob Lee:**

Heh

**Mark Falco:**

please bring burritos

**Cameron Purdy:**

so no one provided feedback. just out of curiosity, any particular worries about just dropping the additional levels of privacy? going back to the simpler model that we started with?

and in the meantime, i’ll “re-run” all of the problem cases through that, just to see if i missed any sharp edges.

**Gene Gleyzer:**

Call?

**Cameron Purdy:**

sure

decision at this point: dropping the additional levels of privacy & going back to the simpler model that we started with (public and non-public i.e. private/protected)

**Mark Falco:**

With no security security between modules in the same "universe"?

**Cameron Purdy:**

there’s no “security” anywhere (that’s part of the beauty of the design)

but access to the private bits of a different class’ object can only occur with the cooperation of that object …

so not as “wide open” as gene may have discussed with you

**Mark Falco:**

Gotcha, so does reflection exist in such a world?

**Cameron Purdy:**

not as a separate concept (in java, it was glommed on), but as an integral concept.

so it is possible to ask a reference what it contains (properties and methods)

**Mark Falco:**

Ok, and you can reflect only on the same stuff you'd be able to "cast" to?

**Cameron Purdy:**

yes, from a Type, you can only see what you can call, and you can only call what you can see

(from a Class, you can see more, but it’s not callable without a reference. generally, you can only get to the Class info from within the module)

from an object o, you can get the Type (it’s a property). inside the object (i.e. with a “this” reference), you can get info like the Class itself, but that’s not available via a public property

looking for a name … a Class, when “bound”, is an Object (or perhaps a reference). a Method, when “bound” (i.e. to a reference), becomes a Function. what is the equivalent w.r.t. a Property? what is the name of the unbound vs. the bound form? (the bound form is something that contains methods such as “T get()” and “void set(T)”, while the unbound form is more like the meta-information about the property)

ok .. i have decided to use “Property” to represent the unbound form, and “InstanceVariable” to represent the bound form

**Mark Falco:**

I was going to suggest Field

**Cameron Purdy:**

which do you prefer?

“Oracle Corporation's (NYSE:ORCL) transition to cloud has been taken care of for the most part, believes Joel Fishbein, analyst at BTIG. According to him, the toughest and most significant part of company’s migration towards cloud computing has been dealt with and the company only needs to tidy up loose ends in order to start full blown operations.”

**Gene Gleyzer:**

No comment

**Cameron Purdy:**

created a hip chat room for language discussion

working on formalizing properties and the built-in annotations thereof.

there’s a new room called “ecstasy”

i posted an updated chunk of properties doc, and the conversation from me & mark that might explain some of it …

**Rob Lee:**

Got to use my favorite two words: "ship it" :-)

**Cameron Purdy:**

Nice

**Mark Falco:**

Congratulations

**Cameron Purdy:**

getting close to nailing down the versioning and “conditional inclusion” (i.e. #ifdef) design. if it comes out clean, it will be a pièce de résistance

Seems to hold water. Each VM Structure (module, package, class, etc.) will have a condition constant (optional). The constants for imported (linkable) items will likewise have a condition constant (also optional). Byte code will have COND/NCOND ops for conditional blocks. Add transitive closure and we're done. I'll test it tonight while I sleep.

This adds any number of dimensions to the compiled module format

**Gene Gleyzer:**

If it's a runtime concept, how is it different from an "if"; optimized?

**Cameron Purdy:**

because it’s knowable at a structural level, not just during the runtime

also because the changes themselves can be structural, e.g. “int foo()” in version 1, “String foo()” in version 2 .. structural items that are naturally incompatible

(the latter answer being far more important than the former. it’s an “if” that affects both the structure of the module itself, as well as the logic inside the curly braces.)

what’s incredibly beautiful about the design that i’m honing in on is that the same conditional structures are used for both the outside and the inside of the curls. so each VM Structure (e.g. a class structure, a method structure) can point to a condition, and that same condition can be pointed to from code (e.g. an “if”)

… and that structure can also define version information (what version this class is, for example) as well as version dependencies (i need to import that module, but i need a version that is at least v2)

comprendez-vous?

implementing a dot delimited integer notation for versioning with arbitrary branching is an interesting take on huffman encoding, methinks

i might just be enjoying myself 😉

VersionConstant

Represent a version number. A version is either a base version, the subsequent version of another version, or an revision of another version. A version number is represented as a dot-delimited string of integer values; for example, version "1" is a potential base version number, version "2" is a subsequent version of version "1", and version "1.1" is a revision of version 1.

For each integer in the version string, the first integer is considered the most significant version indicator, and each following integer is less significant, with the last integer being the least significant version indicator. If the least significant version indicator is zero, then the version is identical to a version that does not include that least significant version indicator; in other words, version "1", version "1.0", and version "1.0.0" (etc.) all refer to the same identical version. For purposes of comparison:

The actual versions vA is identical to the requested version vR iff each version indicator from the most significant to the least significant is identical; in other words, version "1.2.1" is identical only to version "1.2.1".

The actual versions vA is a match for the requested version vR iff each version indicator of the requested version from the most significant to the least significant is identical to the corresponding version indicator in the actual version; in other words, version "1.2", "1.2.1", and "1.2.1.7" are all a match for version "1.2".

The actual versions vA is substitutable for the requested version vR iff each version indicator of the requested version from the most significant to the least significant is identical to the corresponding version indicator in the actual version, or if the first different version indicator in the actual version is greater than the corresponding version indicator in the requested version; in other words, version "1.2", "1.2.1", and "1.2.1.7", "1.3", "2.0", and "2.1" are all substitutable for version "1.2".

actually, i think i can merge the last two (match and substitutable) since the substitute for 1.2.0 is the same as the match for 1.2 in my current definition

so it’s based = (“identical”) and >= (“substitutable”)

s/based/basically

<https://www.youtube.com/watch?v=F1xAUfdK9FE>

as i’m digesting today’s conversation, i’m liking it more and more

regarding the analogy of digesting, it was way too big of a meal, so i will be digesting it for a while

**Gene Gleyzer:**

Yes, there's something very appealing in this "visitor pattern"

**Mark Falco:**

I was thinking the event handlers may need a way to indicate they wish to stop consumption for awhile, something I've felt was missing from the exabus Collector concept

**Cameron Purdy:**

Rob - just so you know what’s going on

we’re talking about how objects are visible across thread boundaries

the plan of record (as of this specific moment in time) is quite a bit different from what i had proposed in the past, although for sake of historical revision, it is different primarily for what we’ve chopped off

specifically, the general case of global visibility of shared mutable data is gone

references can not escape a thread unless the references are thread safe

four cases of thread safe:

1) immutable

2) TLO (thread local, i.e. context local)

3) if the only mutable state is mutable only via CAS (Property extends Ref. Ref has “T get()” and “void set(T)”. Property, or perhaps ThreadSafeProperty adds the method “boolean CAS(T, T)”

4) messaging (a la Erlang) aka “channels” (a la Go), with all values being required to be Threadsafe (recursively defined)

Mark - regarding queueing (e.g. queue depth, behavior on overflow), specifying async-only vs. auto (i.e. sync calls allowed on the same actual thread), pausing the queue, virtual queues (load balanced across multiple instances) etc., these should all be settings inherent to the “channel” type, i.e. when one writes a class as “channel Counter { … }” instead of “class Counter {…}”

(or perhaps, “service” instead of “channel”)

depending on the settings, it could use the “door” threading approach that we discussed

todo: read <https://www.erlang.org/course/concurrent-programming>

<http://learnyousomeerlang.com/errors-and-processes>

**Rob Lee:**

[missing the context you guys have, but...]

@mark - isn't "stop for awhile" a bit of a slippery slope? it implies a contract on producers to buffer (how much?), or it implies acceptable lossless or need for a checkpoint/resync mechanism

**Cameron Purdy:**

Rob - that was the reason for sending that second link .. i.e. the “uh oh” conditions on async programming are old enough that we should be able to pick up the best ideas (and reject the worst) from e.g. erlang

**Mark Falco:**

hey rob, I definitely should have elaborated both in my own head and on the thread.  This would largely be related to flow-control, as an handler may find that it can’t inject more into some output channel and thus need to not input more for the time being, but should also not just sit blocked out onput.  So I’ll reword my request to “what about flow control”.  I suppose there may also be a case for “I can’t do anything more right now, stop processing this channel until someone else reregistered me”, that doesn’t sound like a flow-control use case.

**Cameron Purdy:**

i’m in NH this week, so unless you guys are driving up here, let’s postpone the wednesday discussion

**Mark Falco:**

Enjoy, see ya next week

**Cameron Purdy:**

i’ve been working for 4 hours this morning. i’ve written one paragraph.

(more correctly, i’ve typed pages and pages, but by process of refinement, the net remainder is one paragraph)

do you guys ever look at the hip-chat thing?

**Gene Gleyzer:**

Yes, briefly

**Mark Falco:**

?

**Gene Gleyzer:**

I guess that was a "no" :)

**Mark Falco:**

the only message I had from you was “yes, briefly”

**Gene Gleyzer:**

Can wrote:

do you guys ever look at the hip-chat thing?

**Mark Falco:**

ha

occasionally

**Gene Gleyzer:**

BTW, I hope Rob has his phone on silent - it's rather early out there:)

**Mark Falco:**

he’ll learn

**Cameron Purdy:**

he should get up on east coast time, like the rest of us 😃

oh wait, never mind. i usually get up on west coast time.

trying to put @ro into words … “The second capability is language-level support for non-mutating references to an underlying data structure that may itself be mutable. In other words, it is desirable to be able to obtain a second reference to an object that explicitly omits mutating operations and support for mutable state.”