> Is Smalltalk, or some modern variant of it(Dart?), worth getting into? People gush about it a lot, especially its "environment". Stuff like Dolphin Smalltalk kind of confuses me.

My experience with smalltalk is only with it's descendent objective-c, but I think it's something that's worth "being familiar with" so you are aware of how alternative views on OOP work. For instance, you mentioned you're learning Java, but both it and C++ have a similar view on how OOP works (java more strictly adhering to this than cpp). Unlike both where you have the same general pattern of invoking methods on objects, to my understanding smalltalk (and thus obj-c) has the notion of "sending messages" to objects. This seems like just a semantic difference, but when you start writing code you see that it really influences the way you code in each system. (I'm going to just talk about obj-c below since that's what I'm familiar with, but I assume the same principles will apply to smalltalk as well).

Concretely, you can think of "message sending" in objective-c as having method dispatch always being done at runtime based on the function name. Unlike in C++ where you have a vtable lookup at runtime but the offset to look at is still computed at compile time, in obj-c you can think of it more as a hash table of method name to function implementation. Indeed, you can do things like send *arbitary* messages to objects at runtime, and the object can be coded to handle it even if you haven't defined the method at compile time (think of Java reflection on steroids, and less ad-hoc since it's built into the language and meant to be used). The general "dynamicness" of obj-c leads to a lot of neat patterns and powerful techniques like key-value observing, and it's my personal belief that the dynamicness of obj-c is a great fit for GUI-type applications and is probably why it was chosen during the nextstep days and inherited by osx. (Sadly, Swift basically has none of this and while having a lot of syntactic sugar really lacks some of that elegance).

But going back to smalltalk, I know a lot of people talk about the power of its repl (and how "modern" repls aren't anywhere close to what smalltalk had), etc. so I think it's worth looking into for edification but not necessarily something that's crucial to master. (In the same way one might look into Haskell or Ocaml just to get a taste for pure FP even if you aren't going to be using it as your bread and butter).

(And since TC is the only place I get to share the thoughts in my head, I have a small rant about the semi-official Objective-C successor that never was: FScript. At this point it's an obscure, almost unknown language but

anyone who has programmed in objective-c before should check it out; it has some interesting ideas of combining smalltalk type message passing with APL like array semantics. The result is pretty cool, and with a bit more massaging it could absolutely have been the objective-c "successor" instead of swift. )

> Is smalltalk good for writing cross-platform, standalone executables? Would learning it be especially educational?

I think go or rust are the go-to languages for those if you want ease of cross-platform, on account of their ecosystem of 3rd party packages. (If you only care about the 'nixes and are willing to special-case support for osx at times then C++ could also work, but you'll be spending a bit of time on the build system.)

> I'm learning Java for a class, and while I can see some benefits to the object-oriented way of doing things, I find Java to be exceptionally ugly and verbose. My impression is that Smalltalk is like a "better" Java.

Java is a bit too dogmatic in it's OOP-ness at times, but in terms of practicalness once you learn Java then learning any of the algol-derived languages should be pretty easy. Also it's used very heavily in the industry, but be warned that industry Java is even more horrific than anything you've seen in class. Newer java versions are a bit less verbose though, and with some good 3rd party libraries like Guava it's not really too bad. Maybe try kotlin or scala if you want to leverage the JVM but have a bit more syntactic sugar?