What About TypeScript? A Java Programmers Perspective

Introduction
Is it a déjà-vu, or did I accidentally enter a time machine this morning? A couple of minutes ago I saw a video introducing two brand new features: Intellisense (aka auto-completion) and refactoring. It wasn't even a particularly fancy refactoring – just "rename variable".

And it felt just great!

Believe it or not, even in 2014 there's still a wide range of popular programming languages that simply aren't suited for auto-completion, let alone complex refactorings. Groovy is one of the languages, but it's not a big problem, because the Groovy community is small. But one of them isn't merely popular, it's ubiquitous, so ubiquitous that the lack of refactoring and (good) auto-completion is a real pain: Javascript.

So today I'm going to introduce you to TypeScript, a language trying to address some of the issue of current Javascript while trying to keep the spirit of the language alive.

By the way, part of the déjà-vu impression was to observe which IDE was used to present the all-new Intellisense feature: Visual Studio. Back in 1996 I sort of fell in love with Visual Studio because of it's famous Intellisense!

However, times have changed a lot. In 2014 TypeScript programmers aren't restricted to a Microsoft product. There seems to be decent support by JetBrains WebStorm / IntelliJ IDEA and by Eclipse, just to name a few.

A little disclaimer
At this point I should mention I don't have any hands-on experience with TypeScript. This article is merely a compilation of several blogs, articles, forum discussions and the TypeScript project page itself. So if I happen to get something wrong, please help me to improve this article by dropping me a short comment.

What's it all about?
TypeScript was conceived at Microsoft, in a .NET centric environment. .NET programmers are used to languages designed to support large applications. Javascript, in contrast, hat originally been designed with small programs in mind. At the time nobody expected large applications to be written in Javascript. (Quite the contrary, it was even dismissed as dangerous during a decade). That's why there's no proper type system in Javascript, and why the language lacks a native mechanism to split large applications into smaller packages and modules.

Plus it's a language that alienates programmers coming from the object-oriented (more precisely: class-oriented) world. Javascript's inheritance is based on prototypes – a concept every bit as powerful as the mechanism

based on classes, but looking completely different.

So TypeScript is Microsoft's attempt to address these issues by improving the language in a careful, almost gentle way.

Is it necessary?
I've seen a lot of discussions – sometime even flame wars – whether TypeScript is necessary or not. It offers a lot of solutions somebody already solved in the Javascript world independently. There's Bower, there's require.js, there's ECMAScript 6 and so on. However, I think those discussions miss the point. TypeScript is just there. It's up to you whether you use it or not. I for one haven't decided yet if I'm going to use it. In any case it's an interesting project that might help bring the industry forward.

Actually, I suppose whether you like TypeScript or not heavily depends on your background. Seasoned web designers won't be particularly attracted by TypeScript. Nor will they be attracted by Dart, let alone GWT. They already have their decent tool chain, they've got a lot of experience and some of them even have learned to glorify the shortcomings of their pet language as advantages. Typescript introduces at least one additional step – compilation – that might be considered as a big disadvantage by many web designers.

However, TypeScript should be very attractive to .NET and Java programmers.

Look and feel of the TypeScript language
That's because TypeScript accomplishes to use a syntax that's very familiar to OO programmers. At the same time it manages to be fully compatible to Javascript. So you can use every library of the Javascript world: jQuery, Prototype, Backbone.js, AngularJS, ...

Let's have a look at the source code (my first TypeScript program, actually). The standard example of the text books – complex numbers – shows the idea nicely:

```
1  class ComplexNumber {
2    real: number;
3    imaginary: number;
4
5    constructor(real: number, imaginary:number) {
6      this.real=real;
7      this.imaginary=imaginary;
8    }
9
10   plus(other: ComplexNumber): ComplexNumber {
11     var sum: ComplexNumber =
12             new ComplexNumber(this.real+other.real,
13                               this.imaginary+other.imaginary);
14     return sum;
15   }
16
17   toString(): string {
18     return this.real + " + " + this.imaginary + "i";
19   }
20 }
21
22 var pi = new ComplexNumber(3.1415926, 0);
23 var i = new ComplexNumber(0, 1);
24 alert(pi.plus(i));
```

No matter whether you're a C#, C++ or Java programmer, the syntax should look familiar to you. The most no-

table difference is that the type follows the variable instead of preceding it – just the way PASCAL and Scala do it. The advantage of the : notation is that it's easier to make the type declaration optional. It may look a little odd at first, but you get used to it pretty soon.

Actually, my demo ignores a nice feature of the language: type inference. Neither does the type of the methods nor the type of the variable `sum` has to be defined. You see this at the end of the demo. The TypeScript editor correctly inferred the type of `pi` and `i`, so Intellisense offered me the `plus` method when typing.

TypeScript as a tool to teach Javascript

The TypeScript project offers a nice Playground to experiment with the language features. One particular nice feature of the playground is it's split screen: on the left hand side you type object-oriented, strictly typed code, and the left hand side shows the resulting Javascript code.

Among other things this may be a nice tool to teach OO programmers the prototype approach. The Javascript corresponding to the `ComplexNumber` looks remarkably different to the TypeScript version:

```
var ComplexNumber = (function () {
    function ComplexNumber(real, imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }
    ComplexNumber.prototype.plus = function (other) {
        var sum = new ComplexNumber(this.real + other.real,
                    this.imaginary + other.imaginary);
        return sum;
    };

    ComplexNumber.prototype.toString = function () {
        return this.real + " + " + this.imaginary + "i";
    };
    return ComplexNumber;
})();

var pi = new ComplexNumber(3.1415926, 0);
var i = new ComplexNumber(0, 1);
alert(pi.plus(i));
```

Interesting Features

I already mentioned two killer features: making Javascript a typed language improves tool support, bringing both refactoring and auto-completion to another level. Eclipse tries to do some auto-completion with pure Javascript, but it's a far cry from what it can do if it's backed by a typed language. That's also a problem with Groovy, by the way: I like writing small programs in Groovy, but on the long run I seem to be unable to do without auto-completion. Visual Basic made me an addict, back in 1996 (was it really this late?).

To the relief of many web designers types are optional. I suspect this will become a major problem in the future, but of course optional types are unavoidable if you want to use standard Javascript libraries such as jQuery. That's a problem to more radical approaches, such as Dart.

I already mentioned type infererence. munificent says on reddid TypeScript's type inference was fairly limited. However, I was pretty impressed by the demos in this video by project lead Anders Hejlsberg.

Modules and generic data types are also supported (albeit munificent confuses me by claiming generics are not

supported – maybe they are a recent addition?).

However, generics exhibit a problem already visible in the `ComplexNumber` demo: Type Erasure. TypeScript compiles to native Javascript, so there's no way to support type during runtime. That's an annoying problem, particularly to those guy building libraries and framework. However, it's astonishing how many programmer never become aware of the problem. Obviously most business logic can be implemented without reified generics.

There are even optional parameters and optional variables. However, they are a disappointment: they don't prevent NullPointerExceptions. Don't confuse them with Ceylons optional types. They have a different purpose. They allow you to omit function parameters. But you still have to deal with `undefined` values.

Write Once, Run Anywhere?
Any Browser. Any Host. Any Os.

Does this ring a bell to you? It's an interesting move of Microsoft to – partially – abandon their Windows-centric strategy. Unfortunately the TypeScript home page doesn't display nicely on my iPad, so there still seems to be some work to be done. It's a positive sign, nonetheless. TypeScript applications aren't restricted to the Internet Explorer, they are meant to run on any device.

Open Source
Another interesting – maybe even courageous – move of Microsofts is to put the language under a Apache Licence Version 2.0. The source code is available at http://typescript.codeplex.com/sourcecontrol/latest#README.txt.

Idiomatic Javascript and the risk of adopting TypeScript
No doubt adopting TypeScript is a lot less risky than adopting Dart or GWT (if you're doing Javascript before). The latter two require you to start from scratch, and there's no easy way back. They generate highly optimized Javascript that's typically incomprehensible to the average Javascript programmer. Actually, it's not even meant to be read. It's just meant to run fast.

TypeScript, in contrast, generates idiomatic Javascript. That's Javascript that can be read and maintain by every Javascript programmer.

So they say there's no risk at all. If you're sick of TypeScript, just switch to the generated code and you've got a good old Javascript project.

However, on the long run I wouldn't bet on it. Like every other compiler TypeScript is in competition with other vendors. The pressure to generate optimized code is high. Maybe they'll continue to generate idiomatic Javascript, maybe not.

Nonetheless TypeScript follows a fairly conservative approach, so using it shouldn't pose no big risk to your deadlines.

Compilation times and debugging
Judging from the complains on the internet – or rather, the lack thereof – the compiler seems to be fast enough.

However, TypeScript is still at a very early stage – maybe most programs are still small :).

Debugging seems to more problematic. It's possible to debug TypeScript (instead of debugging the generated Javascript code), but it seems to require some additional steps, such as adding source maps and debugging in the browser (as opposed to debugging in the IDE). On the other hand, most JavaScript programmers are already familiar with source maps.

Conclusion
TypeScript is an interesting addition to the panopticum of attempts to improve Javascript:

- Emscripten (compiles more or less every language to JS)
- GWT / Vaadin (hide JS completely from the Java programmer)
- Dart (tries a take a fresh new start by replacing the JS including its virtual machine)
- TypeScript (tries to improve JS by extending it)
- Coffeescript
- Javascript
- JSLint (assumes there's a subset of JS that can be safely used)
- ASM.js (a very low-level, machine-oriented approach to high performance JS)

As you can see, TypeScript resides somewhere in the middle. It's a conservative, evolutionary approach to make web programming more productive. It also tries to adopt the new ECMAScript 6 standard – aka the next generation of Javascript.

Maybe more important, it provides an easy access to Java programmers. Even more important, it's an easy access to .NET programmers who don't have to abandon the IDE they're used to. So chances are TypeScript may find it's niche.

Further reading
a comprehensive introduction to TypeScript by project lead Anders Hejlsberg
The TypeScript project page
The TypeScript playground
comments of a dart project team member
seven months with TypeScript
Scott Hanselmann on TypeScript
Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem?
OIO's introduction to TypeScript's module system

This entry was posted in Concepts of programming languages, ECMAScript, Javascript and tagged improving JavaScript, TypeScript on February 3, 2014 [http://www.beyondjava.net/blog/typescript-java-programmers-perspective/] by Stephan Rauh.