Applause from Michelé Clarke, Tariq Ellis, and 6 others

Adrian D. Finlay
@thewiprogrammer. Writer @hackernoon. Code, LOTS of it. Mangos, LOVE THEM! Barbering. Health. Travel. Business. & more! Network w/ me @ adriandavid.me/network
Oct 1, 2017 · 22 min read

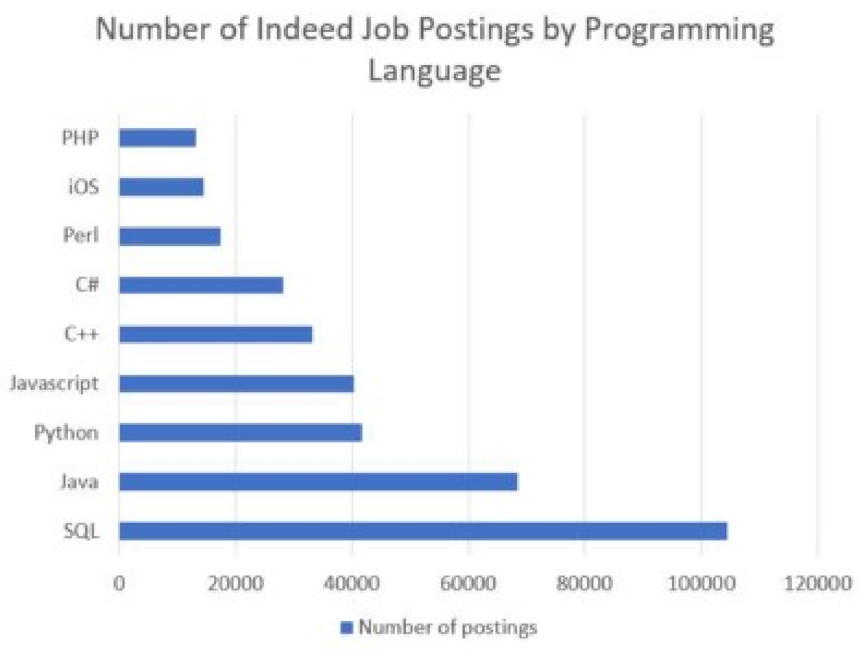
How To Become a Java Web Developer (2017)—Part I: Java Web Development Rudiments

Java Banner [10]

Somehow you've stumbled into programming and gained interest in the Java Programming language. Maybe it was because of school. Or maybe you saw the Java logo on your desktop. Perhaps you figured out that **Java is almost everywhere**—from Blu Ray software to Android devices via the openJDK. You may have noticed that Java routinely ranks as the #1 (or among the top) most popular programming language according to the TIOBE Index and that 6 billion devices use Java [1][2]. You may have also noticed that Java is the second most popular programming language to JavaScript (excluding SQL, which is a query language) according to Stack Overflow[35].

According to the BLS: Software Developers raked in an average of $102,280/yr and $49.17/hr in 2016 and demand is growing at a rate of 17% from 2014 into 2024; By 2020, there will be 1.4 million CS related jobs and only 400,000 CS graduates with the necessary skills. **Java is commonly one of said skills.**

Peradventure you've done some research and discovered that ~**70,000 job postings** (a +30,000 increase from 2016) on the popular job site Indeed.com included the Java programming language as a skill in 2017[2] .



[3] — codingdojo.com

Curious as you are, you may have noticed that 3/4ths (depending on the source, 65%–85%) of Java developers work on **web or enterprise applications** and that 90% of the Fortune 500 use Java on the server-side[4]—a claim stated by Oracle. Indeed, it's an excellent time to be an Enterprise Java developer.

Browsing through the job boards you may have noticed a theme—keywords like "J2EE", "Spring", "Servlets", "Hibernate", and the like. You may have asked yourself—"How do I get from here to there?". Perhaps you're enticed by reports of above average compensation packages for Java Enterprise developers or dreams of designing the "next big thing", or both.

Maybe you've done a Google search and have only found John Thompson's article on the topic [5]. Maybe you haven't asked any of these questions: all you want to know is **how to become a Java web or enterprise developer.** Either way, **the answer is—there is no single definitive answer**. The skills you will need will ultimately depend on the specific role you are seeking. Furthermore, the skills you may generally find Java roles demanding may vary based on a bevy of various factors, such as region and industry, among others. The truth is (and I might be vilified for this) that **you don't need to have every skill listed on a job advertisement** (even among those marked required!). Many employers hire individuals based on aptitude, ability to learn quickly, and, to put it bluntly, because they like them[6]. Despite this, **skills still matter—the work doesn't get done without them.**

Having said all this, perhaps the question to start with is: **"Which are the common core skills of Java Enterprise & Web Developers?"**. The skills discussed will describe the skills of engineers who work on the full software stack. However, as engineers work on various parts of the stack—front end, back end, EJBs, etc.—one may read the portions relevant to them.

It is important to note that this article will not discuss the interview and pre-interview aspects involved in acquiring a job in java development and will instead focus on the core skills of Java Enterprise Developers. To dig into that material, I overwhelmingly recommend **Cracking The Coding Interview by Gayle Laakman McDowell**. You will not be disappointed!

It is also important to note also that this article will focus on technical skills as opposed to soft skills. While soft skills are **invaluable**, they fall out of the scope of this article. However, it is worth noting that good organization, communication, and interpersonal skills are key. **Check out Soft Skills by John Somnez** and **The Clean Coder: A Code of Conduct for Professional Programmers by Robert C. Martin**.

Without further adieu, let's take a peek.

Impatient? Here's a condensed sneak peek: http://bit.ly/2gBUBRJ If the bit.ly link expires or throws a 404, comment on the article, and I will provide a new hyperlink.

Got Suggestions? Hate the article? Love the article? Anything in between? **Let me know in the comments.**

Java—This one is fairly obvious.

> *"The Java Programming Language is a is a general-purpose computer programming language that is concurrent, class-based, object-oriented,and specifically designed to have as few implementation dependencies as possible."[7]*

Java is designed to *Write Once & Run Everywhere*, a notion otherwise known as **WORE**. This is accomplished through the use of executing *Java bytecode* on the *Java Virtual Machine*, which is itself implemented on several platforms. We won't spend much time here, as it should be obvious to the reader that one needs to know the Java language to be a Java developer (duh). **The point to grasp** is that a serious Java developer should **thoroughly understand and aim to master all the elements of the Java programming language itself** and core features as well as important packages (including but not limited to): java.lang, java.util, java.concurrent, java.io.

Why I Personally Like Java: You can develop robust, scalable, trusted**,** tested, reliable **Desktop** (JavaFX, Swing), **Mobile** (Android), **Web & Enterprise** (JEE, Spring, Struts) solutions.

Keep Track of Java Evolution: Java 9 is (as of publication) slated to be released in 17 days on September 21. Will it actually meet it's release date? You have to keep up to date to find out. Don't get left behind the curve—Java 9's modules will fundamentally change how we approach Java programming. Consider attending (or at least follow online) the Java One Conference hosted by Oracle every year during early October. You should also follow a magazine or online publication such as DZone / Java Zone, which features articles talking about the latest and greatest in Java Development.

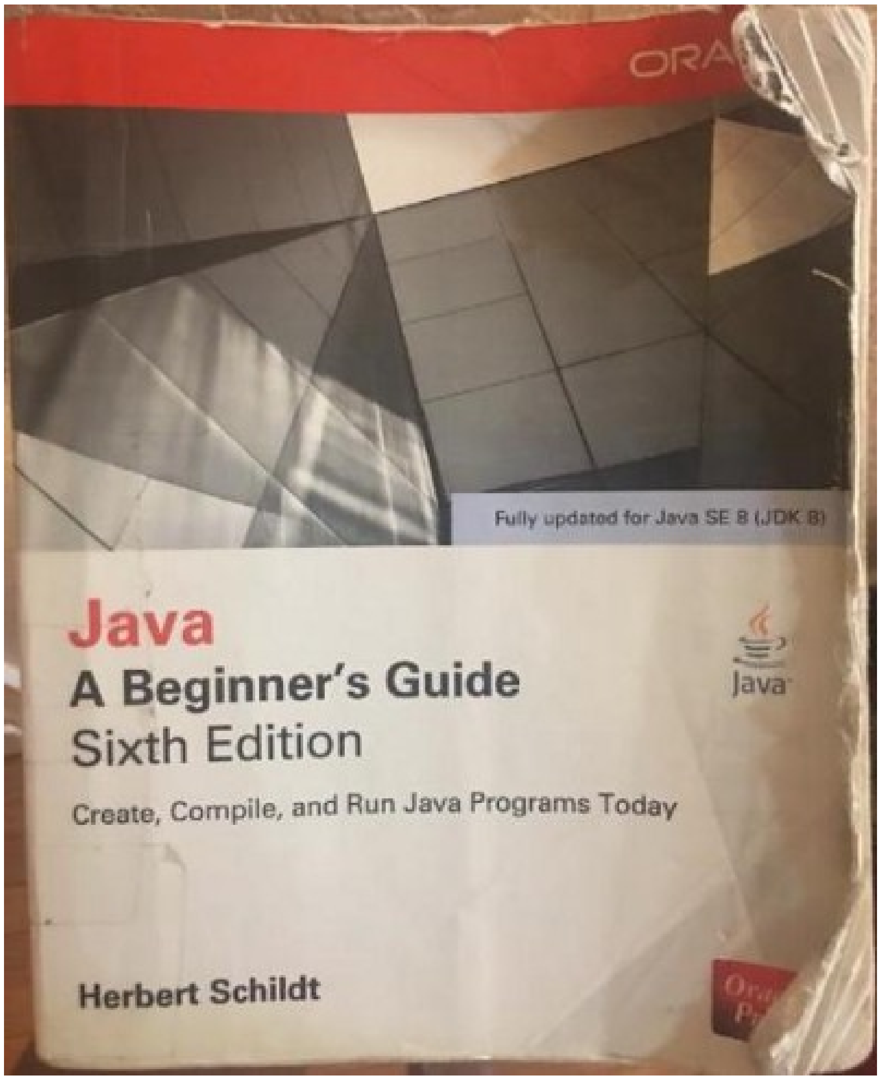
UPDATE: Java SE 9 / Java EE 8 met it's release date! Release notes for Java SE are here. Release notes for Java EE are here.

Where to learn, recommended resources:

Programming is something best learned by both understanding the tools you are working with as well as plain old writing code, to be blunt. My personal recommendations for learning Java are:

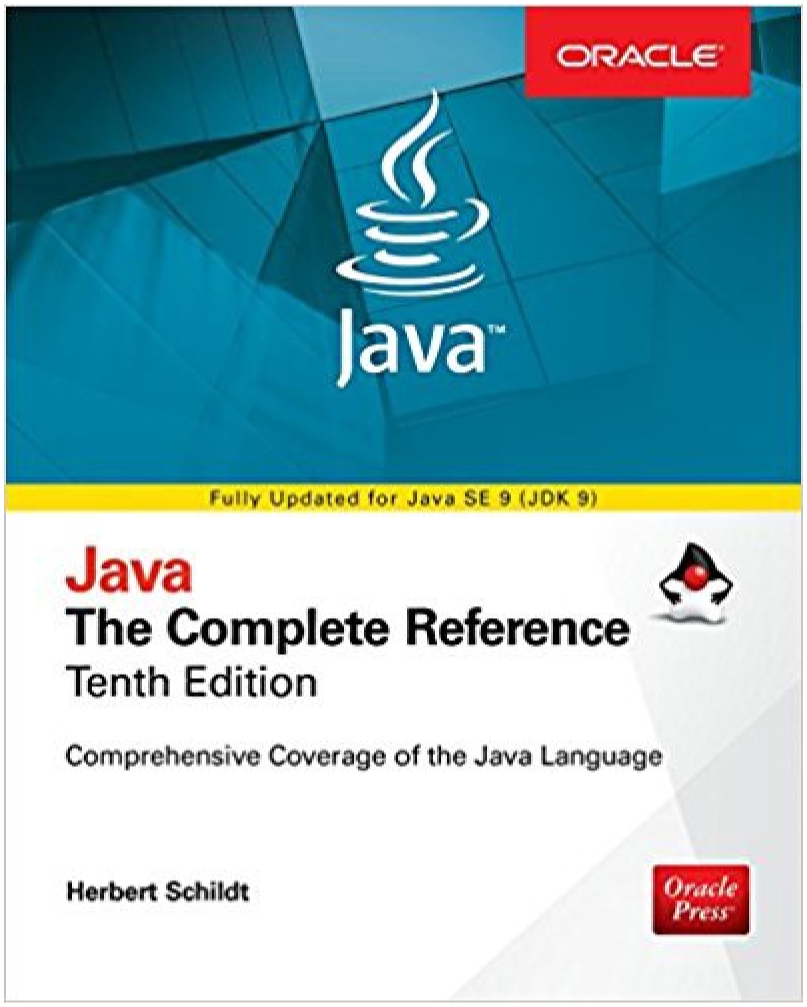
Java: A Beginner's Guide, 7th Ed. (Oct. 13. 2017 release date) —The 6th Edition to this book was my Introduction to Java. I came to Java after having learned C++ in school. The details seem all to fuzzy. I remember the Fall 2014: I would sit in my IST 302 class wishing the material were about Java instead of Project Management, and I loved Project Management very much & still do. This book is my number one resource for beginner's. I have personally spoke to the Author, Herbert

Schildt, thrice, and have zero question in my mind that the 7th edition will be just as excellent. Published by the Oracle Press. Covers Java 9.


ORA
Fully updated for Java SE 8 (JDK 8)
Java
A Beginner's Guide
Sixth Edition
Create, Compile, and Run Java Programs Today
Java
Herbert Schildt

My tattered, worn personal 6th Ed. Copy. Affectionately known as my second love.

Java: The Complete Reference, 10th Ed. (Oct. 13. 2017 release date) The next logical step after reading the Beginner's Guide. Jam packed with critical information. **Compulsory reading.** Also written by Herbert Schildt and published by the Oracle Press. Covers Java 9. **I personally believe that all java programmers should own this text, with no exception.** I own the 9th Edition of this text.


ORACLE
Java
Fully Updated for Java SE 9 (JDK 9)
Java
The Complete Reference
Tenth Edition
Comprehensive Coverage of the Java Language
Herbert Schildt
Oracle Press

Java, The Complete Reference, 10th Ed.—Herbert Schildt

Core Java: Volume I, Fundamentals, 10th Ed.
Let it be known: After reading this text, there is no way you could tell Cay S. Horstmann that he doesn't know his stuff. This text reeks of the Author's very deep and thorough knowledge of the Java Programming Language. Goes beyond the language basics and leaves the user satisfied. Covers Java 8.

Core Java: Volume II, Advanced Features, 10th Ed.
The next logical step after reading the First Volume. Every bit as satisfying. Covers Java 8.

Qualifications—How to say you know what you know

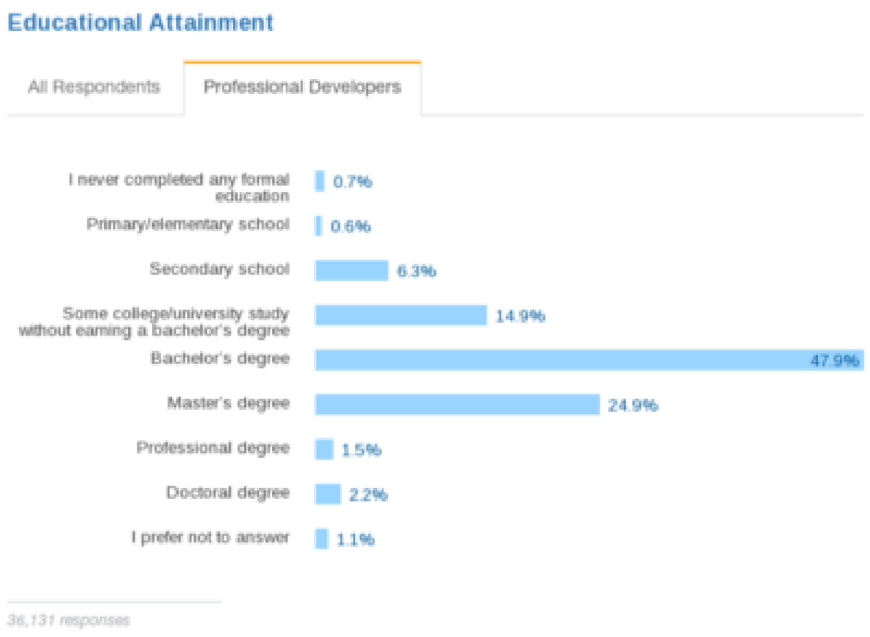
This one is a bit tricky. The most common qualifications of software developers are: **college degrees, high-school diplomas, technical**

certifications, and **technical bootcamp completion certificates.** Qualifications are important, in my opinion, because they are a standardized, mostly-reliable way of estimating a person's level of competence and knowledge of a particular domain.

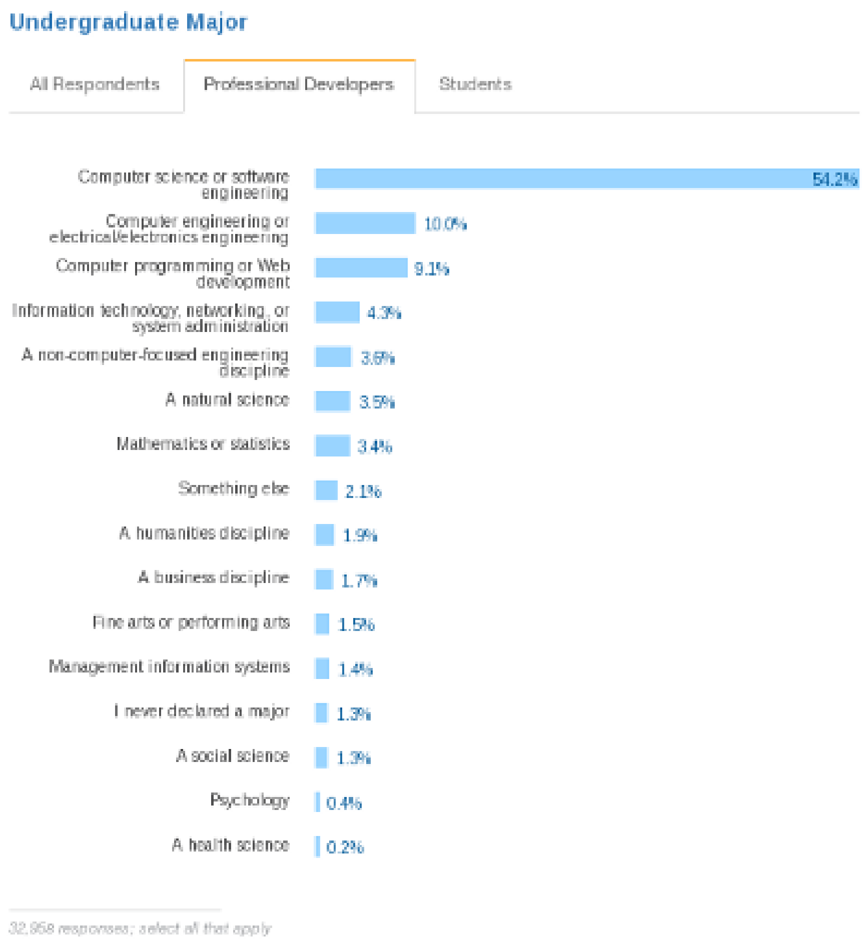
> ***Formal Education:*** *Degrees & Diplomas*

According to Stack Overflow (the most popular programming community on the web and the most popular programming informational site on the internet [36]), **76.5% of professional developers hold a bachelor's degree or higher, 14.9% had some college experience, 7.6% had below college education,** and 1.1% preferred not to answer. Slightly over half (~56%) of people who took this survey provided a complete response to the following:

Stack Overflow Developer Survey 2017—*Educational Attainment* [35]



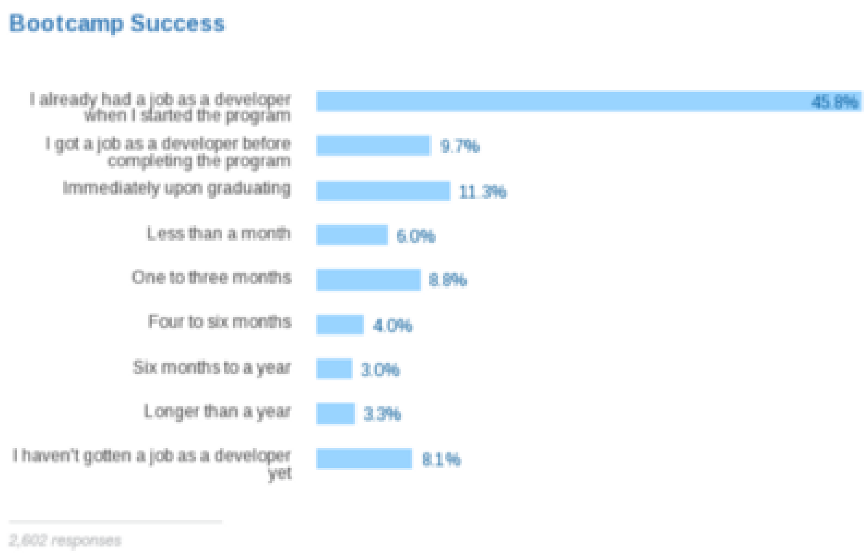
Of those who had studied at a college or university, **54.2% studied Computer Science or Software Engineering**, 24.9% majored in a closely related discipline, and 20.9% focused in non related fields[35].



Miscellaneous Highlights of the Responses by Professional Developers on the Education portion of the Stack Overflow Survey to various questions:

- 67.9% said formal education was at least somewhat important.
- 91.1% of developers reported being at least partially self-taught.
- 9.4% of developers used bootcamps. 22.8% recommend coding bootcamps.
- **16.4% sought industry certifications**.
- Several others used on-the-job training, online courses, open source contributions, hackathons, coding competitions, & part-time courses as a means of education.

Bootcamp Success among Software Developers [35]



The Stack Overflow Developer Survey 2017 data is very interesting and you can find it here. You may regard it as a SOTU of sorts, for the developer community. I strongly recommend reading their annual reports.

> *What does this all mean?*

There are a number of conclusions (and indeed a number of data sources from which to draw said conclusions) one can make from the available data to try to determine the importance of qualifications to the software engineering profession. I won't make them for you, but a few things stand out to me. These are my opinions.

- While you do not need a degree to be employed as a Software Developer, **the majority of developers seem to hold degrees**. Try browsing the indeed software developer listings for your local area—I'm willing to bet that you will find as I found that it all depends. Some organizations state a degree as a requirement while others state is at a preference. Your mileage truly does vary, but I willing to guesstimate that the majority will at least prefer a degree. **Even if they do state that they require a degree, apply anyway**. I will again be vilified for this, but at the end of the day a company is looking to do business with you—if you can make them money, they may be happy to hire you. **It's a dirty yet oft-repeated secret that companies may hire people who don't have all the requirements**. You've got nothing to loose. The worst they could do is not hire you.

- **There are several qualifications that employers recognize when hiring Software Developers**. On average, an immediately relevant degree seems to me to be the most prominent qualification above bootcamps and closely related degrees. While this may change, this trend seems to be holding fast, for now at least.

- Perhaps the most astonishing findings—among professional developers who did seek a degree, **slightly over a fifth didn't study a relevant topic**, **slightly over a fourth studied a closely related field**, and the most astonishing**—only 54 studied either CS or Software Engineering!** What this makes clear to me is that **your college major** is probably **not that important when being considered for a developer job**. I wish Stack Overflow would separate Computer Science from Software Engineering for the 2018 Survey, however, as the two curricula are definitely not the same.

- That 91.1% of developers are self-taught is indicative of the fact that **real world work is simply beyond the topics taught in the classroom**. The classroom incontrovertibly has it's use. However, if you want to be a developer, you **most certainly will never be able to rely on the classroom alone to give you all the tools you need**. You will also **not be able to afford not continuing to learn**. Period.

- **Advanced degrees are not as important as in many other fields**. ~48% of professional developers have just a bachelor's degree. While the data alone may not fully support the claim that advanced degrees are not as important, I think the fact that the data shows that the vast majority of professional developers (69.2–70.3%) lack a degree past a bachelor's is telling. Take a browse through the job listings—how often do you see a Master's Degree (or higher) as a requirement? I rest my case.

> ***This is all fine and dandy but… which qualification should I get?***

At the end of the day, I can't tell you which option is best for you.

Some **Colleges & Universities** can be devastatingly and exorbitantly expensive, to the ire of many people. At a top ranked CS institution (I won't say where) between Tuition, Fees, Room, Board, Books, &

Supplies,etc. could cost as much as $70K USD per year. **That's equivalent to someone's annual salary and higher than the median household income in the United States.** Should you finish school in 4 years that would run you about $280K. Should you take out loans for school, you would pay more than that, obviously, due to interest. The decision is a personal one—one is tasked to determine for their own how much they are willing to pay for a degree as much as they'd be willing to pay for a car, or anything else. My thinking is to look on the **return on investment—how much money are you in a position to earn because of the particular degree you achieved**.

On the other hand, financial aid is available at many schools and low cost education exists. **Degrees are the longest standing and most marketable qualification, by far.** A college degree provides you the flexibility to more easily leave software development should you later decide to. Also, a college degree is critical to obtaining a higher-ranking or managerial job. If thought about carefully, a college education can be a very rewarding investment. After all, bachelor's degree graduates are reported by some to earn well over a million more than their non bachelor degree holding peers [37]. I think there are many explanations for this that make sense. However, I do not believe that the correlation of incomes and bachelor's degrees is evidence alone of cause (degree) and effect (money). Notwithstanding, a college degree is valuable and is a solid option. I do not, however, recommend a for-profit or non-accredited college or university.

In my opinion **Certifications** are useful and I really like some of them, but they will not give you as much employ-ability as would a university degree. Nevertheless, they can potentially be a good (In my opinion) qualification to make you stand out of the pack for a targeted area, like Java. **Check out the Oracle Certified Associate & Programmer in Java Programming**, here. Oracle makes several strong claims about the industrial value of their Java certification and I think very positively of it. I have met software engineers for which this certification has benefited them and I have seen the certification asked for in a few job postings.

I was previously not a fan of **Coding Boot-camps** and had largely dismissed them as a scams but my mind has changed. After-all—Programming is not a gimmicky tricky that can be learned in a mere 16 weeks? That was my former thinking. While I still don't believe that one can become a full fledged developer in 16 weeks, I think a lot of progress can be made. I think Boot-camps can be useful if the boot camp has strong industry connections, if the individual is self-motivated and learns outside of the boot-camp, and if the individual posts their code on the web for organizations to see. Check out Gayle Laakman McDowell's (author of *Cracking The Coding Interview*, among other books) article on the matter: So that whole coding bootcamp thing is a scam, right?

Self Taught. Usually these individuals hold a high-school diploma or GED. In my opinion, many of these individuals wind up more knowledgeable about their area of study **because they had the freedom to focus on said area**. All things considered, I personally believe that all serious developers have to be self-taught to a very significant degree, and that seriously self taught people are very commonly passionate. After all—willingly applying the discipline needed to become a Software Developer without school holding your hand is hard work. If you take it upon yourself to do all the dirty work of discipline and discovering what it is you need to know to become a developer, then immediately I know that you take software development **very seriously. Aside:** CNN Money reported than in 2010 about 38% of web developers had less than a four-year college degree, according to the US Census.

You should check out Cory Althoff's Self-Taught Programmer's group ;)

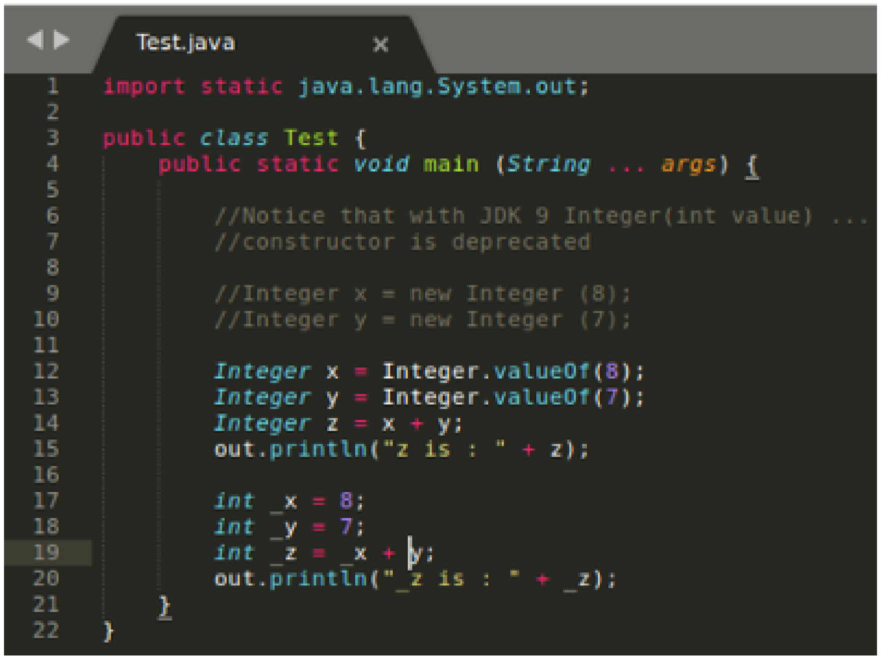
Fundamental Computer Science—Core Foundations

> *"Computer Science is the study of computers and computational systems. Unlike electrical and computer engineers, computer scientists deal mostly with software and software systems; this includes their theory, design, development, and application." [38]*

Whether you are self-taught or you graduated from University, you should have a handle on core fundamental computer science. There are several reasons why you should understand the fundamentals of Computer Science. Instead of listing them, I will try to summarize them all with one statement—You need to understand: **what exactly it is that you are doing in coding, what exactly it is that you are coding on, and how exactly it works**. Can you skate by without knowing CS fundamentals well? Maybe, maybe not. Should you try? **Definitely not**. The situation is loosely akin to playing basketball while blindfolded. You may have some success, you may score a basket, but you **can't really see so as to properly navigate what it is you are doing.** You should understand what it is that you are actually doing when you compile, link, and then execute a program. All the way down to ones and zeros, and beyond.

Perhaps Basketball isn't your taste or that analogy didn't quite work the way I wanted it to. Take for example, driving: It's useful to know when you are driving how the car works—but you can get away with not knowing many of the details, often, but not always, when all you need to do is to drive. The situation with programming is a little more subtle than that. When all you need to do is build an application, you might be able to get away with not understanding how the compiler does what it does. However, the subtlety is that **programming requires a little more understanding of the underlying phenomena than driving does to perform the task well**. Understanding and refining how you use and design data structures and algorithms is key to just about everything—from security to performance and everything else.

You should understand why the first method of integer addition is less efficient than the second method of addition in the following:



```java
import static java.lang.System.out;

public class Test {
    public static void main (String ... args) {

        //Notice that with JDK 9 Integer(int value) ...
        //constructor is deprecated

        //Integer x = new Integer (8);
        //Integer y = new Integer (7);

        Integer x = Integer.valueOf(8);
        Integer y = Integer.valueOf(7);
        Integer z = x + y;
        out.println("z is : " + z);

        int _x = 8;
        int _y = 7;
        int _z = _x + _y;
        out.println("_z is : " + _z);
    }
}
```

Another thing: Did I need to use the **varags**? Nope, absolutely not. In actual reality the JVM doesn't notice the difference—it still sees an array—but you should understand the difference.

And so on.

Side Note: In the process of crafting this example I learned that the Integer constructors are deprecated in Java 9 in favor of static methods! You **never** stop learning in programming. For those curious, you can find out more, here.

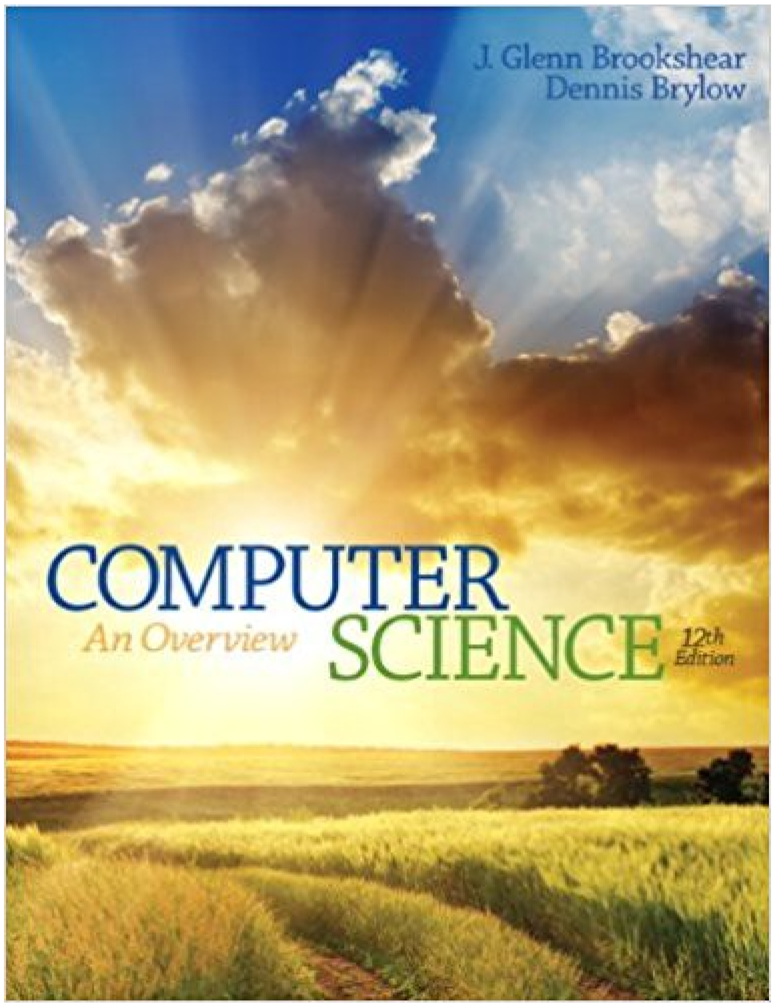
According to the **Computing Sciences Accreditation Board**, a board that includes folks from the Association for Computing Machinery (ACM) and the IEEE Computer Society, there are for foundational areas to Computer Science [38]:

- The Theory of Computation
- Algorithms & Data Structures
- Programming Methodologies and Languages
- Computer Elements & Architecture

It is beyond the scope of this article to discuss in depth the foundations of Computer Science. However, I will point you in the right direction. See below.

Where to learn, recommended resources:

Computer Science: An Overview, 12th Ed.



While there may be many texts on several different topics concerning Computer Science, this text was among the only ones I could find that discussed as broad an over view of the fundamental ideas of computer science. While this book may not provide you with everything you need to know, it will equip you with starting knowledge to work with when thinking about Computer Science.

The Theory of Computation—Chapter 12
Algorithms & Data Structures—Chapters 5 , 8
Programming Methodologies and Languages—Chapter 6
Computer Elements & Architecture—Chapters 1 , 2

SQL—The world is all about data

> *"Structured Query Language—SQL is a domain specific language used in programming to manage data held in a relational database management*

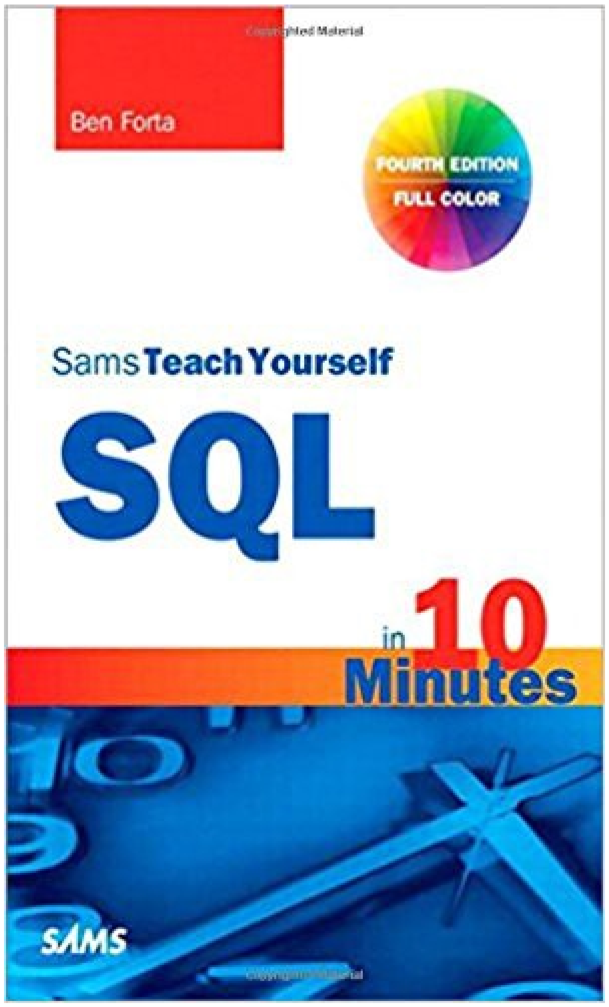
> *system (RDBMS), or for stream process a relational data stream management stream (RDBMS)."[8]*

If you're going to be doing back-end Java development, you'll likely be dealing with relational databases in some fashion; you will need to understand SQL. In addition to learning SQL, it's safe to say that **one is pretty much required to learn a RDBMS platform such as MySQL or PostgreSQL**, as ANSI SQL can only get you so far. The current edition of ANSI SQL is SQL:2016.

According to Rebel Lab's Java Tools & Technologies Landscape 2016 Survey [27], the following RDBMSs have the leading mind-share among Java Developers: Oracle DB with 39%, MySQL entrenched with 38%, & PostgreSQL with a strong 29%. It is important to note that respondents could choose more than one RDBMS.

Where to learn, recommended resources:

Sam's Teach Yourself SQL in 10 Minutes a Day ~ Ben Forta
This text should be considered compulsory reading for application developers. At 288 pages, it is reasonably digestible for a week's reading. You will not be able to grab a DBA job after this book, but it doesn't matter, because you're not trying to be a DBA, you're trying to be a Java Enterprise Developer. It helps that this book is one of the most widely (if not #1) most printed text on SQL a testament to it's quality and consequently Forta is widely revered for his work. At present, Forta's text is in it's 4th Edition which covers SQL:2011 and SQL:2016 is upon us. Despite being one version behind, SQL:2011 is the most popularly implemented version as of present and the updates to the SQL standard (SQL:2016) are available freely online[11]; Most of them concern JSON. Because it is compact, concise, and technically sound, I still recommend reading this book.



Sam's Teach Yourself SQL in 10 Minutes—Ben Forta

Tutorial's Point MySQL Tutorial
MySQL is a very popular and very powerful common open source RDBMS. MySQL was made very popular in part by the popularity of LAMP (Linux/Apache/MySQL/ Perl, Python, PHP) stacks of the late 1990's and early 2000's. You can read more about the history, here.

OOP—Grouping together the "do" & the "stuff"

> *"It is dangerous to make predictions, especially in a discipline that changes so rapidly, but one thing I can say with confidence is that I have seen the future, and it is object-oriented"—Grady Booch, Father of UML, 1996 [9]*

Object Oriented Programming (OOP) is a programming paradigm by which an application is structured in terms of object behavior and interaction. In OOP, an object is an abstraction that binds attributes and subroutines. Commonly, contemporary OOP parlance refers to attributes as fields or data members and subroutines as methods. Class based object design is the most common contemporary approach to OOP.

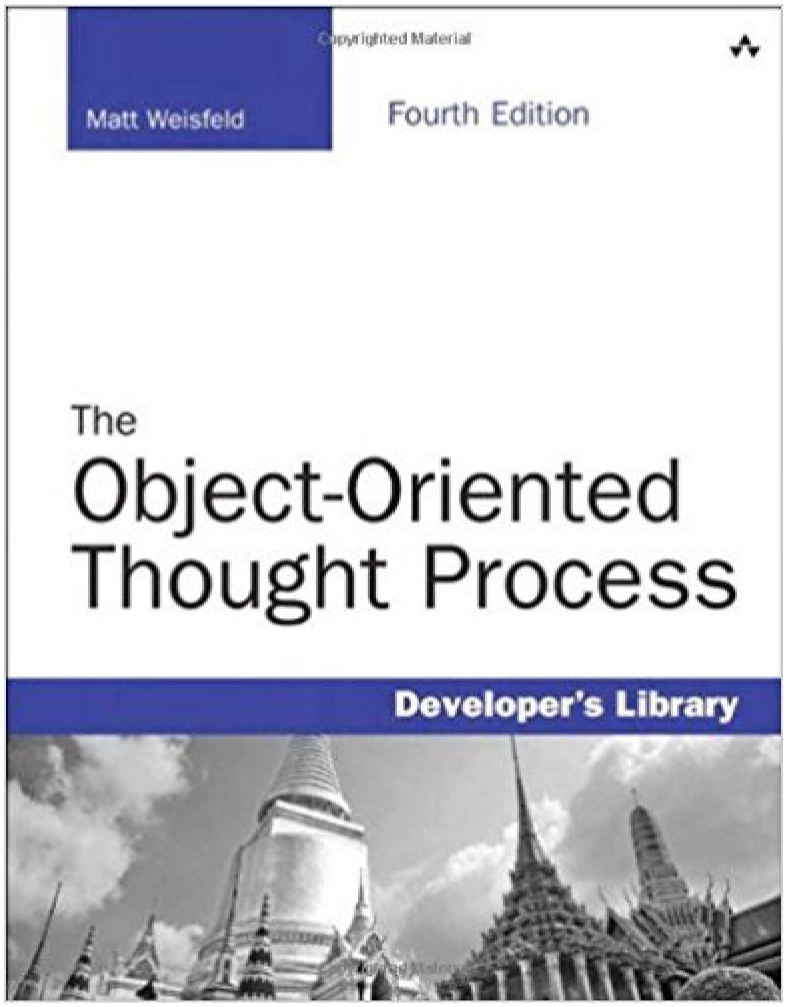
Most of the popular and modern languages: Python, Java, C#, C++, Ruby, PHP, Scala, etc. all include support for Object Oriented Programming as a core component of their programming model. The fact is—a contemporary developer will probably not be able to circumvent the Object Oriented Paradigm (unless you're into weird stuff like the programming language "Brainf***", which already has developers extending it to be object oriented!). Get the point? OOP is ubiquitous.

Key to the architectural design of Object Oriented Software is the **Unified Modeling Language (UML)**. UML is a common means by which Software Developers describe and design the way objects are to behave in an Application. "The Unified Modeling Language (UML) is a graphical language for OOAD that gives a standard way to write a software system's blueprint. It helps to visualize, specify, construct, and document the artifacts of an object-oriented system. It is used to depict the structures and the relationships in a complex system."[42] UML is not restricted to Object Oriented Design (OOD), however.

The despicable truth: Most of us will probably not go out of our way to grab a generic Object Oriented Programming book (phew!, I said it). Instead, we will probably learn Object Oriented Programming while learning a programming language or through a university course. However you do it, **you must do it**, as this skill is borderline quintessential to the field. You will be a better developer for it. You will thank yourself for it. Your boss will thank you for it, and so will the future maintainers of your code ;) .

Where to learn, recommended resources:

The Object-Oriented Thought Process, 4th Ed., Developers Library
My personal recommendation. This text is written in a language agnostic way (though primarily C#), with examples in several languages available online from the publisher. I really like the way this book is written and I have not seen something (as modern at least) written quite like it.



Head First Object-Oriented Analysis and Design
Although I am not a personal fan of the Head First series (just give it to me raw, forget the kiddy stuff), many people are. This text is very popular and I have confidence that you won't be steered wrong. To be technically accurate: This book is more concerned with Object Oriented Analysis than the last text which is not exactly the same as Object Oriented Programming. Likewise, Object Oriented Design does not necessarily mean programming, and as such *Your Mileage May Vary* when considering this book for OOP specifically. That being said, although the philosophical approach to the topic differs, they concern the same and similar material.

Some Notes: While book reading is a great approach, you may also find benefit from taking a course, like I did. This is the approach by which (in my estimation) the majority have learned OOP.

Tutorials Point Unified Modeling Language(UML) Tutorial

Tutorials Point Object Oriented Analysis & Design Tutorial

While OOP is the dominant paradigm, **Functional Programming** is experiencing a resurgence in popularity and you should definitely be aware of it. This influence has been felt community wide and example such as 1) Clojure, Scala, Jaskell, 2)Emergence of functional interfaces, lambda expressions, and method references in Java 8, 3)Emergence of Functional Support in Spring 5[12], among others, testify to this.

HTML, CSS, JavaScript—Visual Representation of web pages & their behavior

> *"HTML is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript it forms a triad of cornerstone technologies for the World Wide Web.* ***HTML specifies the content of web pages, CSS specifies the presentation of web pages, and JavaScript specifies the behaviour of web pages.****"[13]*

This trio represents the cornerstone core technologies involved in what is called *front end programming.* In addition to learning these languages you should also considering learning a View/MVC/MMVM framework such as AngularJS, Angular 2/4, or ReactJS + Redux, Vue, etc. The inclusion of one of these frameworks greatly simplifies development for many projects, relieving you from having to become bogged down in every detail of HTML. Today's Java Developer should be familiar with at least one of these groundbreaking frameworks.

Where to learn, recommended resources:

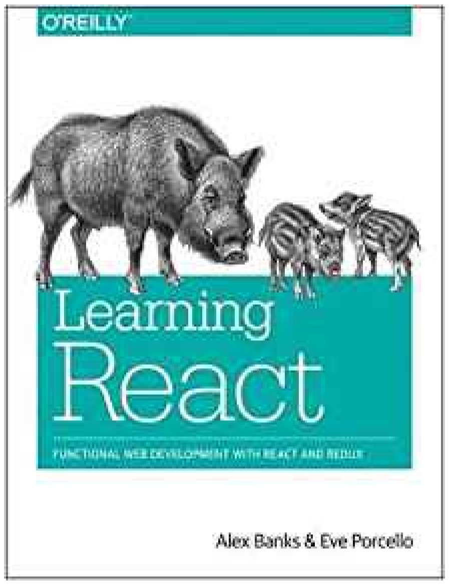
HTML5
https://www.tutorialspoint.com/html/

CSS3
https://www.tutorialspoint.com/css/

JavaScript
https://www.tutorialspoint.com/javascript/
JavaScript: The Good Parts

JavaScript Framework—ReactJS
https://www.tutorialspoint.com/reactjs/
https://facebook.github.io/react/tutorial/tutorial.html
Learning React

Learning React: Functional Web Development with React and Redux


O'REILLY
Learning React
FUNCTIONAL WEB DEVELOPMENT WITH REACT AND REDUX
Alex Banks & Eve Porcello

JavaScript Framework—Angular 4(Angular Platform 2+)
https://angular.io/docs
ng-book: The Complete Guide to Angular 4

A special note on Angular:
TypeScript—An ECMAScript superset that compiles to JavaScript (consequently all JavaScript programs are valid TypeScript programs). TypeScript notably adds **classes** among several different language features.

AngularJS/Angular 1.X—The original Angular Framework. Written in JavaScript.

Angular/Angular 2.X—A complete rewrite of AngularJS. Written in TypeScript.

Angular 4—The next version in development of Angular2, Backward Compatible with Angular 2. Written in TypeScript.

All Three: HTML, CSS, JavaScript
https://www.amazon.com/Web-Design-HTML-JavaScript-jQuery/dp/1118907442

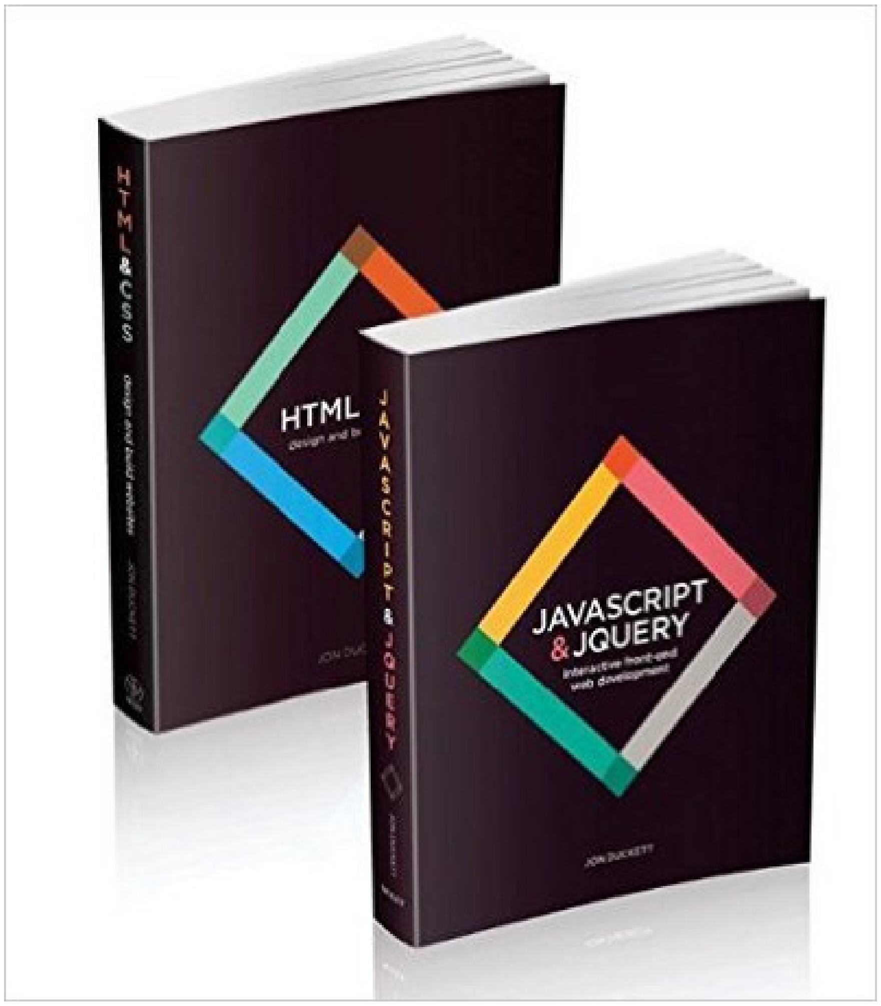
While Duckett's text was published in July of 2014, and evolution, particularly in the CSS and JavaScript communities has led to newer editions of both languages, this text is still relevant. The reason for that is largely due to version adoption. According to the W3Schools[21], **ECMAScript 5 (2011) is the only fully supported version by all major web browsers**, while ECMAScript 6 (2015) is partially supported and ECMAScript 7 (2016) is poorly supported. Similarly, although CSS4 is upon us, many browsers have still not fully supported CSS3.

For example, the results of css3test.com running on my Win10 x86–64 box,

Opera Browser v. 46**–58% CSS 3 Support**
Google Chrome Browser v.60**–58% CSS 3 Support**
Google Canary Browser v.63**–59% CSS 3 Support**
Google Chromium Browser v.60**–58% CSS 3 Support**
Firefox Browser—**67% CSS 3 Support**

As of: 09/16/2017.

Quality work takes time. Consequently, this book is still very much relevant.

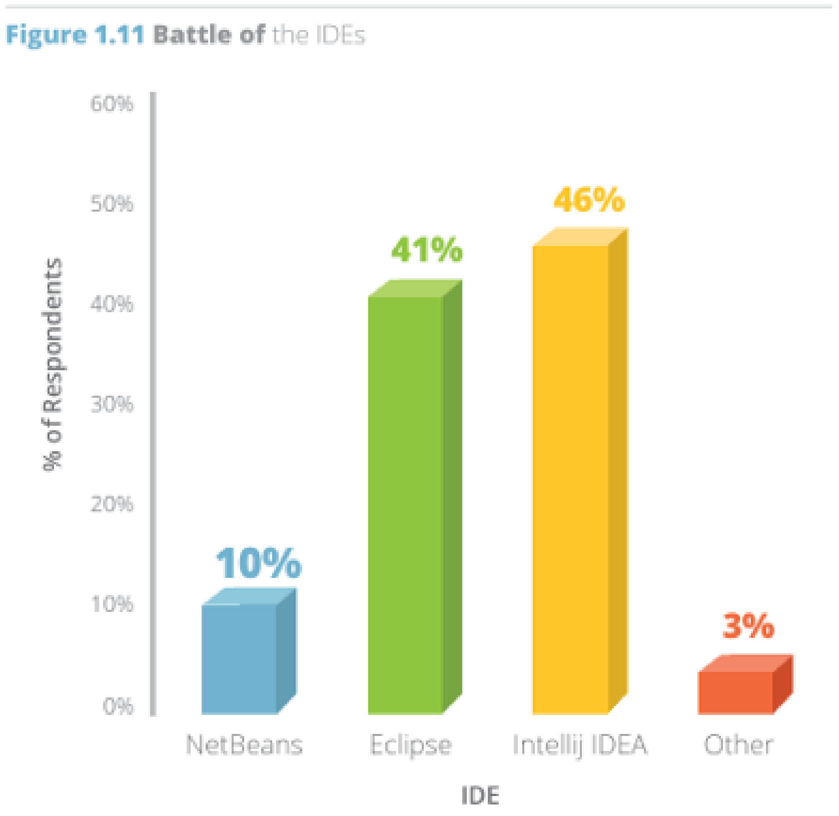

HTML & CSS
design and build websites
HTML
JAVASCRIPT & JQUERY
JAVASCRIPT
& JQUERY
interactive front-end web development
JON DUCKETT

Web Design: HTML, CSS, JavaScript—Jon Duckett

Integrated Development Environment—The Developer's Workdesk

The Java Environment is rich with tools. Among them, **Integrated Development Environments(IDE)** are standard tools that all Java engineers need understand. While it is not necessary to use an IDE to build software, IDEs dramatically increase productivity by integrating the various common tools used in building software into one environment. Among a whole host of productivity tools include: a debugger, utilities for managing dependencies, facilities for plugins, auto-complete, automation tools, templates, compilers, and many more. Practically speaking, real world development demands that you understand how to use and IDE.

IDE Mindshare among Java Developers [27].



IDE Mindshare [27]

There are several IDEs available and they provide varying support for various languages and technologies. My personal favourites happen to be **IntellIJ IDE** and **NetBeans IDE**. The company who develops IntelliJ, JetBrains, has in particular had unique influence in the Java community. In addition to producing a very popular IDE, the Android Community has adopted IntelliJ as the basis of the Android Studio IDE. Additionally, Google now officially supports Kotlin, a JVM language that is source-to-source compatible with Java, for Android Development.

This publication is the first of a 3-part series. Should you prefer to read the full article it is available here.

Source: ChocolateVanilla.com

The next article in this series, "How To Become a Java Web or Enterprise Developer—Part II: Enterprise Fundamentals, Data, & The Web", is available here.

The works cited for the publication is available in the last article in the series.

Interested in Java? Join my Java group on Facebook:

Join My Java Facebook Group

Interested in Java? Check out my Facebook Group: Java Software Development Group!

medium.com

Like my Content? Subscribe to my mailing list:

This embedded content is from a site that does not comply with the Do Not Track (DNT) setting now enabled on your browser.

Please note, if you click through and view it anyway, you may be tracked by the website hosting the embed.

Learn More about Medium's DNT policy

Don’t forget to give it a.... ;)

IEmoji.com