CI346 - Programming Languages, Concurrency & Client Server Computing (Assignment 1 - John Vos)

by John Vos

Submission date: 14-Jan-2020 04:02PM (UTC+0000)
Submission ID: 118309814
File name: CI346_-_Assignement_1_John_Vos.docx (229K)
Word count: 3375
Character count: 22707

A comparison of two programming languages that occupy different programming paradigms.

Assignement 1:

Programming Languages, Concurrency & Client Server Computing

John Vos

Contents

Major differences between Java & JavaScript:

Programming Paradigms Java vs JavaScript:

Java is an Object-Oriented Programming language (OOP) that is based in "objects". Moreover, these objects can hold properties, functions and or features that belong to them, as well as, holding and or sharing the data it holds. In contrast, JavaScript is a scripting language as well as a functional programming language. Instead of using objects, the language uses functions to achieve certain deliverables. JavaScript is considered a multi-paradigm language as it can be used as OOP, procedural or functional programming language. Java is primarily used as a "back end" language and JavaScript is used as a "front end" language.

Syntax & features:

type checking:

JavaScript falls into the dynamically typed language category; resulting in variables only being known and bound once the code has finished compiling. Java falls into the statically typed language category; resulting in variables being allocated to a "type" as soon as it is declared, not once it is compiled. Type in this instance referrers to if the variable being declared is a string, int, bool, etc... Therefore, in JavaScript the variables "type" is only allocated once the program has finished compiling and in Java the "type" is declared immediately. Resulting in less runtime errors in Java making the compilers job easier as errors are caught before the program is executed. Moreover, dynamic typed languages are considered to not be secure enough due to the type being checked at the run-time.

inheritance:

JavaScript does not support multiple inheritances due to each object being a prototype that can point to another object that it inherits attributes from. One prototype can only connect to one object and therefore cannot support multiple inheritances. In contrast, Java does support inheritance via a subclass and a superclass; which is referred to as the child and parent respectively. The subclass inherits from a superclass (Appendix A).

function overloading:

Java has 3 main ways of overloading methods / functions, these are; the number of parameters, the data types of parameters and the sequence of data types of a parameter (Appendix B). JavaScript however, does not support function overloading natively, as adding function with the same name but different arguments will be considered last defined function.

multithreading:

Java does support multithreading as it is able to execute multiple threats in one instance. JavaScript cannot support multithreading as execution of the code is done within the browser and browser interpreters are single-threaded. However, even though multithreading can improve a program run time, it can also result in a more complex program meaning a harder time trying to debug any issues. Moreover, the chances of incurring a deadlock are increased.

closures:

Java does not support closures; however, it does not rely on them as it is not a functional programming language (unlike JavaScript). Java 8 lambda expression can only access the final variables of an enclosing scope and therefore does not compile; i.e. Java saves values of free variables only to enable them to be used inside lambda expressions and the compiler will avoid creations of incoherent scenarios, therefore, this will limit variable types that are usable inside of lambda expressions as well

as anonymous classes to only the final ones and risks not compiling (Appendix C). In comparison, JavaScript functions and lambda expression do use the concept of closures as new functions of the enclosing scope will maintain pointers where they have been defined (Appendix C).

Compilation & Execution:

Java is platform independent, due to the language being compiled & interpreted in bytecode before execution in the Java Virtual Machine (JVM). In contrast, JavaScript's interpretation takes place in the browser & therefore is written in accordance to browser specification. Resulting in code not being executable on other browsers.

Features provided to enable *modularity* and *separation of concerns*?

Modularity concerns:

Java:

Java 9 has introduced a "new module system" to make modularity simpler and more approachable. This new module system means that modules can; export or strongly encapsulate packages and express dependencies from other modules explicitly. Moreover, modularity has been tackled by demarking clearly defined boundaries between the interacting modules and dependent modules. Modules that are defined must therefore declare any dependencies held in other modules explicitly. This system can therefore verify any dependencies in three phases; during compilation, during linking and during run time, as a result, eradicating any problems of missing dependencies before any issues that could lead to the application crashing. In addition, Java code can hold an interface module that should be kept separate from the implementation module. This therefore allows a client of said module to write and type check against the same interface to ensure any implementation adheres to its own specification. Moreover, Java supports modular programming at both class level and package level and uses package mechanism to provide support for modularity above class level.

JavaScript:

Due to JavaScript being a language used in browsers and browsers not supporting the function of a file being accessed by another for security reasons, modularity cannot be achieved in the same manner as Java. The only way to achieve modularity in JavaScript is through external resources, such as; Node JS or JavaScript libraries / webpack's. JavaScript does also have a standard syntax for importing and exporting modules. Therefore, in its current state JavaScript modularity is dependent on the browsers compatibility to import and export statements. For example, chrome does support import and export, but internet explorer does not. The importance of this falls under the scope of the development of a large web-based application and it being supported by all / multiple web browsers. For example, if a web page requiring JavaScript is developed using a modular approach due to the scope of the project, the web page will not work on some web browsers and this is a grave issue for users.

Separation concerns:

Java:

Java being an OOP language mean that it can separate concerns into architectural design patterns and objects. This can also be achieved by minimising coupling, this is achieved by reducing, minimising and eliminating the complexity of necessary relationships and the law of Demeter. This law states that a unit should only have limited knowledge of other units, i.e. only closely related units to the unit in question and a unit should only talk to its units its related to (friends / close friends) and not others

(strangers). Moreover, inversion control can be achieved through factory patterns, service locator pattern, strategy pattern, dependency injection, contextual lookup, template method pattern.

JavaScript:

JavaScript handles the separation of concerns is through module management. This is useful as a module manager creates a standard module API under which all other modules can use, such as, tinyapp. Therefore, they can focus solely on the service they provide and keeps them lightweight. Moreover, application such a tinyapp can aid by launching asynchronous calls early on in the runtime which will speed things up. Another method is through events emitters as these provide a method for communication between various components or application modules. For example, in HTML5 a common event are clicks, i.e. when a user clicks something; this is where event emitters come in to play. Moreover, event aggregators can be used as they collect events from multiple sources via a central object and then develops registered subscribers to deliver. These aggregators can eliminate specialised mediators that keep listeners and emitters decoupled.

What features are provided to enable *code reuse*?

Java:

Java being an object-oriented language means that a key feature for code reuse is composition, this is achieved when using primitive types, as they are automatically initialized to zero, but references are initialized to null. If methods are called for any initialized primitives an exception will be returned. This is important as it enables users to print primitives without exceptions being thrown, moreover, as the compiler doesn't simply just create objects by default for all references due to this creating unnecessary overhead in most cases (Appendix D). Another integral part of code reuse is through inheritance, the Java syntax for inheritance must be specified with the keyword "extends"; moreover, as stated previously inheritance works through a super and a sub class (Appendix A). Finally, if a user ensures that functionality is moved out of class instance methods, changes non primitive input parameter types to interface types and chooses less-coupling input parameter interface types their code (in general) will be more reusable.

JavaScript:

JavaScript having function programming language features means that arrays and objects can reuse the initial state and avoid side effects (e.g. updating global variables inside another scope) by using immutable and avoiding mutables (when possible). Another method is through functions, invoking the functions, setting parameters inside the function and returning values (Appendix D).

Prevention of *corruption of shared data* between concurrently executing threads/ tasks?

One key feature Java has to avoid corruption of shared data when working with the execution of concurrent code is a method called multithreading. Java does so in multiple ways, two mechanisms available are extending thread classes and the implementation of runnable interfaces (Appendix F). Another method is keyword synchronisation, this method synchronises code and locks it internally on the class or object. This is to ensure that a single thread is being executed in regard to the synchronised code. Furthermore, this is achieved by unlocking and locking resources before threads can enter synchronised code, i.e. resources that can lock threads are unlocked. Moreover, a user can either completely synchronise methods or just synchronise one block. Another thing to note is while an object is locked, methods that are synchronised and codes are blocked. On the other hand, methods that have not been synchronised are able to continue without obtaining the lock. Therefore, it is required that any methods relevant to shared resources are synchronised; e.g. if a variable requires

synchronisation all methods tied to that variable must be synchronised, or else, methods that have not been synchronised proceed without having obtained a lock first, and as a result, variables state may be corrupted. This can be achieved by using commands such as *"wait();"*, *"notify();"* and *"notify All();"* (Appendix F).

JavaScript does not support multi-threading as it is an interpreted language whereas Java is a compiled language; i.e. JavaScript is interpreted in the actual browse itself in a single thread format. Existing web page could incur vast concurrency issues if JavaScript were to be run concurrently in a browser (such as, Chrome), moreover, components are separated inside the browser into separate processes. JavaScript does hold features such as "setTimeout" can be used to schedule tasks, but this does not truly fall under concurrency.

Java vs JavaScript *Input / Output* and *functional purity* differ?

Java I/O steams: input-output

Java uses I/O streams that aid when performing input-output operations, these streams support all types of characters, files, data-types and objects to fully execute I/O operations. Furthermore, java handles input data via data given to the program using *"system.in"* as the standard input stream, this function takes any standard input from a device, such as, input from a keyboard. Javas standard output stream is *"system.out"* which is used to produce result(s) from a program on an output device, such as a computer screen (Appendix G). The systems output is represented as data received from program in the form of a result and stream which represents the flow of data or the sequence of data.

Java can read input data from the console in 3 ways:	Java can write outputs to the console in 2 ways:
- bufferedreader - stringtokenizer - scanner	- print(String) - write(int)

Java stream can be split into two primary classes subject to the type of operations. Firstly, input sources such as arrays must be taken in for input steams that are used to read data. Secondly, output streams are used to write data into an array. Input & output sources however are not limited to arrays, they can also be used for ByteArrayOutputStream, BufferedOutputStream, etc. Moreover, these streams can be split into two primary classes relevant to the type of file and these can further be broken up into further classes (Appendix H).

JavaScript: input-output

JavaScript handles data input by using window object methods and event handlers. These come in the form of button objects, form objects, options and select objects. JavaScript handles output depending on code given and therefore will display respectively to the code. There are 4 main ways of handling output (Appendix H);

- *document.getElementByid(id)* (innerHTML) is used to access elements, this defines html content
- *document.write()* which is used for testing
- *window.alert()* which displays content using alert boxes.
- *console.log()* which is used to debug.

Functional Purity:

In functional programming a pure function at its core is a piece of code that returns the same result to the one given in the same argument (deterministic) and does not cause any observable side effects. Java is not a functional language and uses I/O (Input / Output) and therefore is inherently impure. However, some functions can perform I/O whilst still being pure if operation sequences are modelled explicitly as both argument and result. Furthermore, I/O operations must also fail when and if the sequence of input is no describing what operations are actually taken since execution begun. Moreover, Java is not a first-class object which is a requirement for functional programming paradigms, however, the nearest to this is in Java Lambda Expressions. In JavaScript functional purity can be achieved using mapping and though not directly, has the capabilities of being a functional programming language. Not all code in JavaScript can be pure however.

Fit for purpose: Java vs JavaScript?

In regard to web development, as it currently stands JavaScript is the go to language, especially compared to Java. Moreover, JavaScript can be directly imbedded inside HTML and therefore holds the ability of implementation through libraries or as a framework, whereas Java requires a Java applet. JavaScript also holds the advantage server side two-fold; one, environment runtimes are optimised and the developer's level of competence. This is due to a plethora of developers and investment to optimise quality and performance of JS tools and engine used in tandem to the language. Moreover, JS holds a framework called TypeScript which supersets JS with class support and static typing, meaning JS infrastructure can be re-used in deployment. Java on the other hand is used primarily in android apps, software products such as Apache storm, finance programs on the server and client side as its fast and reliable, trading applications such as murex, big data programs; i.e. apps. To summarise, JavaScript is used more for scripts related to client-side tasks; such as, jQuery, backbone.JS, AngularJS. Java is used on development that's server-side; such as, Apache, Geronimo and JBoss.

Reflection on implications of findings to be:

Java being an OOP language means that data structures can become objects that are manipulatable in order to create relations between different objects. This means objects can be re-used easily, objects can hide information that should not be accessible and therefore prevents errors. Moreover, Java's OOP approach can make programs more organised and aid in pre-planning. Legacy code being modernised and maintained is relatively simple.

https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/
https://berb.github.io/diploma-thesis/original/092_progtrends.html
https://www.thesoftwareguild.com/faq/difference-between-java-and-javascript/
https://www.seguetech.com/java-vs-javascript-difference/

Appendix:

Appendix A: Java Inheritance

Inheritance: Superclass

Superclass.java | Subclass.java

```java
public class Superclass {
    protected String inherited_superclass = "parent class";

    public void inherit() {
        System.out.println("this is the superclass");
    }

}

```

Inheritance: Subclass

Superclass.java | Subclass.java

```java
public class Subclass extends Superclass{
    private String inherited_subclass = "subclass";

    public static void main(String[] args) {
        Subclass inherit = new Subclass();
        inherit.inherit();
        System.out.println(inherit.inherited_superclass + " inheriting from " + inherit.inherited_subclass);
    }

}

```

Inheritance: Output

```
this is the superclass
parent classinheriting from subclass
```

Appendix B: Method Overloading

4 Overloading: Different number of parameters in argument list

Overloading Display:

```
public class OverLoadingDisplay {

    public void display(char c) {
        System.out.println(c);
    }

    public void display(char c, int num) {
        System.out.println(c + " " + num);
    }
}
```

Overloading Sample:

```
class Sample {
    public static void main(String args[]) {
        OverLoadingDisplay obj = new OverLoadingDisplay();
        obj.display('a');
        obj.display('a', 10);
    }
}
```

Output:

```
a
a 10
```

4 Overloading: Difference in data type of parameters

Overloading Display 2:

```
public class OverLoadingDisplay2 {

    public void display(char c) {
        System.out.println(c);
    }

    public void display(int c) {
        System.out.println(c);
    }
}
```

Overloading Sample:

```
class Sample2 {
    public static void main(String args[]) {
        OverLoadingDisplay2 obj = new OverLoadingDisplay2();
        obj.display('a');
        obj.display(5);
    }
}
```

Output:

```
<terminated> !
a
5
```

Appendix C: Closures

Java: closures

Java_Closures.java

```java
public class Java_Closures {

    void function() {
        int var = 42;
        Supplier<Interger> lambdaFunction = () -> var;      //error
        var++;
        System.out.println(lambdaFunction.get());
    }
}
```

JavaScript: closures

JS:

```javascript
function makeSizer(size){
    return function() {
    document.body.style.fontSize = size + 'px';
  };
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);

document.getElementById('size-12').onclick = size12;
document.getElementById('size-14').onclick = size14;
document.getElementById('size-16').onclick = size16;
```

HTML:

```html
<p>Some paragraph text</p>
    <h1>some heading 1 text</h1>
    <h2>some heading 2 text</h2>

    <a href="#" id="size-12">12</a>
    <a href="#" id="size-14">14</a>
    <a href="#" id="size-16">16</a>
```

CSS:

```css
body {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 12px;
}

h1 {
  font-size: 1.5em;
}
h2 {
  font-size: 1.2em;
}
```

Output:

Some paragraph text

some heading 1 text

some heading 2 text

12 14 16

Appendix D: Class Re-Use

Java Composition:

Java Composition: University

```
public class University {
        private String address;
        private long post_code;
        private int number;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public long getPostCode() {
        return post_code;
    }

    public void setNumber(long post_code) {
        this.post_code = post_code;
    }

    public int getID() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }
}
```

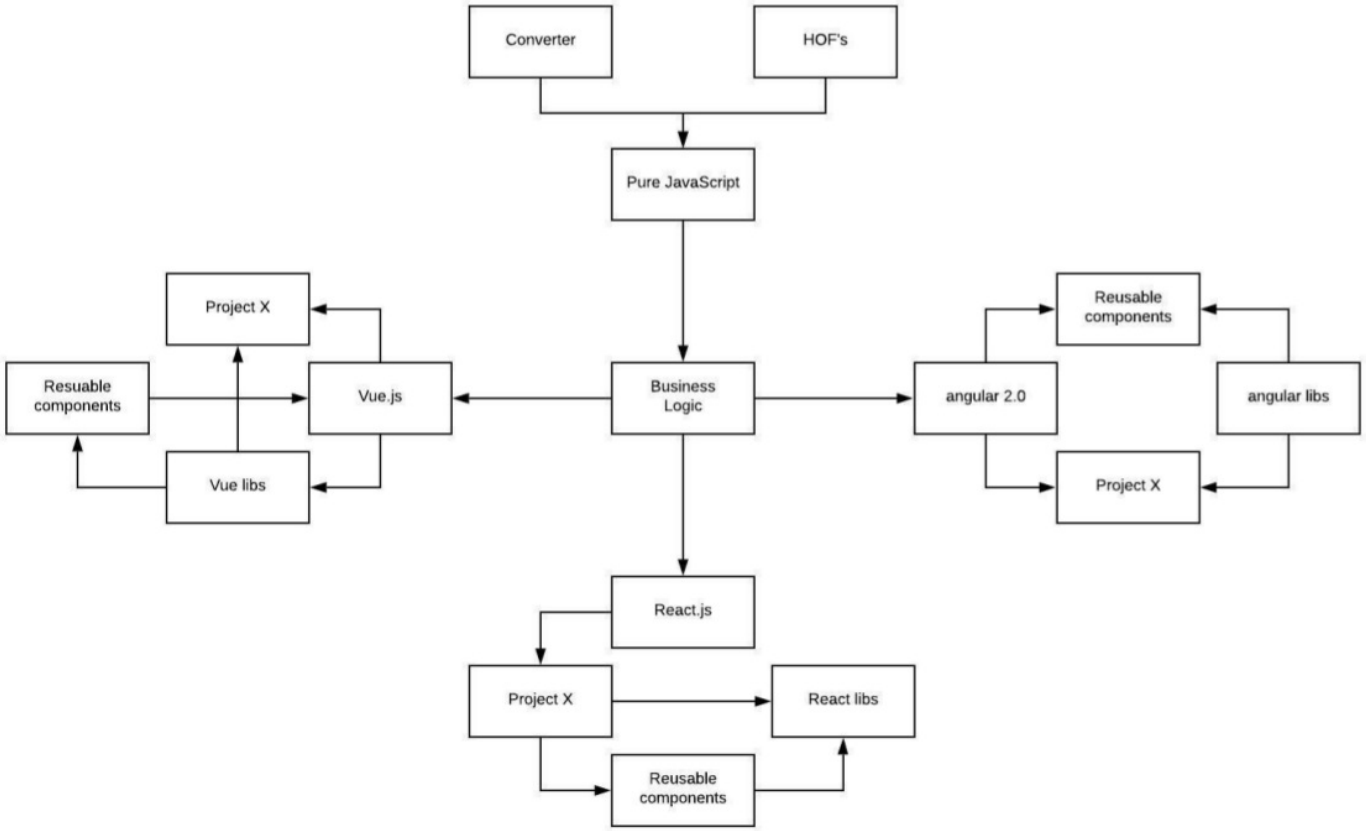
Java Composition: Student

```
public class Student {
    private Student student;

    public Student() {
        this.student = new student();
        student.setPostCode(06410);
    }

    public long getPostCode() {
        return student.getPostCode();
    }
}
```

Java Composition: Test

```
public class TestStudent {

    public static void main(String[] args) {
        Student student = new Student();
        long address = student.getAddress();
    }
}
```

JavaScript:


Converter
HOF's
Pure JavaScript
Project X
Resuable components
Vue.js
Vue libs
Business Logic
Reusable components
angular 2.0
angular libs
Project X
React.js
Project X
React libs
Reusable components

Appendix F:

Java:

https://www.journaldev.com/1061/thread-safety-in-java

JavaScript:

https://www.geeksforgeeks.org/multithreading-in-java/

https://www.ntu.edu.sg/home/ehchua/programming/java/J5e_multithreading.html

Appendix G:

Appendix H:

Java:

JavaScript:

https://www.geeksforgeeks.org/javascript-output/

https://www.w3schools.com/js/js_input_examples.asp

References

Anon., 2011. *HTML Goodies.* [Online]
Available at:
https://www.google.com/search?q=html+goodies+java+vs+javascript&rlz=1C1GCEA_enGB884GB884&oq=html+goodies+java+vs+javascript&aqs=chrome..69i57.17357j0j4&sourceid=chrome&ie=UTF-8
[Accessed 12 2019].

Anon., 2018. *Stack Overflow.* [Online]
Available at: https://stackoverflow.com/questions/46261778/how-interfaces-in-java-used-for-code-reuse
[Accessed 12 2019].

Anon., n.d. *Developer.* [Online]
Available at: https://www.developer.com/java/other/modularity-in-java-java-9-modularity-versus-prior-versions.html
[Accessed 12 2019].

Anon., n.d. *Edcuba.* [Online]
Available at: https://www.educba.com/java-vs-javascript/
[Accessed 12 2019].

Anon., n.d. *The Software Guild.* [Online]
Available at: https://www.thesoftwareguild.com/faq/difference-between-java-and-javascript/
[Accessed 12 2019].

Anon., n.d. *W3Schools.* [Online]
Available at: https://www.w3schools.com/java/java_inheritance.asp
[Accessed 12 2019].

Anon., n.d. *W3Schools.* [Online]
Available at: https://www.w3schools.com/js/js_input_examples.asp
[Accessed 12 2019].

Anon., n.d. *Wikipedia.* [Online]
Available at: https://en.m.wikipedia.org/wiki/Programming_paradigm
[Accessed 12 2019].

Berb, n.d. *GitHub.* [Online]
Available at: https://berb.github.io/diploma-thesis/original/092_progtrends.html
[Accessed 12 2019].

Chaitanya, S., 2014. *Begginers Book.* [Online]
Available at: https://beginnersbook.com/2013/05/method-overloading/
[Accessed 12 2019].

Contributors, M., 2019. *Developer Mozilla.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules
[Accessed 12 2019].

Contributors, M., 2020. *Developer Mozilla.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures
[Accessed 01 2020].

Demeter, 1987. *Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Law_of_Demeter
[Accessed 12 2019].

Diehl, D., 2015. *Seguetch.* [Online]
Available at: https://www.seguetech.com/java-vs-javascript-difference/
[Accessed 12 2019].

Eckel, B., n.d. *Linuxtopia.* [Online]
Available at:
https://www.linuxtopia.org/online_books/programming_books/thinking_in_java/TIJ308_001.htm
[Accessed 12 2019].

Eckel, B., n.d. *Linuxtopia, Re-using Clases.* [Online]
Available at:
https://www.linuxtopia.org/online_books/programming_books/thinking_in_java/TIJ308.htm
[Accessed 12 2019].

Eduardo, 2010. *StackOverflow.* [Online]
Available at: https://stackoverflow.com/questions/1326071/is-java-a-compiled-or-an-interpreted-programming-language
[Accessed 12 2019].

Elliot, E., 2014. *O'Reilly.* [Online]
Available at: https://www.oreilly.com/library/view/programming-javascript-applications/9781491950289/ch05.html
[Accessed 12 2019].

Elliott, E., 2016. *Medium.* [Online]
Available at: https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-pure-function-d1c076bec976
[Accessed 12 2019].

Gadzinowski, K., 2018. *Developers.* [Online]
Available at: https://www.toptal.com/software/creating-modular-code-with-no-dependencies
[Accessed 12 2019].

Gimeno, A., 2019. *Log Rocket.* [Online]
Available at: https://blog.logrocket.com/node-js-multithreading-what-are-worker-threads-and-why-do-they-matter-48ab102f8b10/
[Accessed 12 2019].

Halliday, L., 2018. *Leigh Halliday.* [Online]
Available at: Leigh Halliday
[Accessed 12 2019].

Jenkov, J., 2018. *Tutorials Jenkov.* [Online]
Available at: http://tutorials.jenkov.com/java-functional-programming/index.html
[Accessed 12 2019].

Jiang Li, S. R. M. L. Y. L., n.d. Comparative Studies of 10 Programming Languages within 10 Diverse Criteria. *Department of Computer Science and Software Engineering,* pp. 1 - 139.

Kappert, L., n.d. *Java-Design_Patterns.* [Online]
Available at: https://java-design-patterns.com/principles/#minimise-coupling
[Accessed 12 2019].

Lipp, M., 2019. *Dev.* [Online]
Available at: https://dev.to/rlxdprogrammer/an-overview-on-different-programming-paradigms-functional-style-of-programming-in-modern-languages-3ppo
[Accessed 12 2019].

Mannino, M., 2017. *D Zone.* [Online]
Available at: https://dzone.com/articles/java-8-lambas-limitations-closures
[Accessed 12 2019].

Martin, S., 2019. *Towards Data Science.* [Online]
Available at: https://towardsdatascience.com/java-vs-javascript-which-is-the-best-choice-for-2019-a41ee2d3f78d
[Accessed 12 2019].

Mather, J., 2001. *Java World.* [Online]
Available at: https://www.javaworld.com/article/2077470/java-tip-107--maximize-your-code-reusability.html
[Accessed 12 2019].

McFadden, D., n.d. *Study.com.* [Online]
Available at: https://study.com/academy/lesson/modular-programming-definition-application-in-java.html
[Accessed 12 2019].

McManis, C., 1996. *Java World.* [Online]
Available at: https://www.javaworld.com/article/2077313/code-reuse-and-object-oriented-systems.html
[Accessed 12 2019].

Morel, A., 2019. *Sabe.* [Online]
Available at: https://sabe.io/classes/javascript/functions
[Accessed 12 2019].

Niyaz, 2009. *StackOverflow.* [Online]
Available at: https://stackoverflow.com/questions/39879/why-doesnt-javascript-support-multithreading
[Accessed 12 2019].

Pal, K., n.d. *Mr Bool.* [Online]
Available at: http://mrbool.com/modularity-in-javascript/28646
[Accessed 12 2019].

Pankaj, 2013. *Journal Dev.* [Online]
Available at: https://www.journaldev.com/1325/composition-in-java-example
[Accessed 12 2019].

Pankaj, 2018. *Journal Dev.* [Online]
Available at: https://www.journaldev.com/1061/thread-safety-in-java
[Accessed 12 2019].

Patro, N., 2018. *Codeburst.* [Online]
Available at: https://codeburst.io/functional-programming-in-javascript-e57e7e28c0e5
[Accessed 12 2019].

Paul Bakker, S. M., n.d. *O'Reily.* [Online]
Available at: https://www.oreilly.com/library/view/java-9-modularity/9781491954157/ch01.html
[Accessed 12 2019].

Reddy, A., 2018. *Tutorial Point.* [Online]
Available at: https://www.tutorialspoint.com/What-is-function-overloading-in-JavaScript
[Accessed 12 2019].

Riarawal99, n.d. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org/javascript-output/
[Accessed 12 2019].

Rodrigues, M., 2018. *Vanilla.* [Online]
Available at: https://blog.vanila.io/handling-concurrency-with-async-await-in-javascript-8ec2e185f9b4
[Accessed 12 2019].

Santos, P., 2018. *Medium.* [Online]
Available at: https://medium.com/agrotis-developers/improving-your-javascript-code-reusability-with-functional-programming-paradigms-3a17b8a1ae4a
[Accessed 12 2019].

Shankar, S., 2019. *Concurrent-tasks.js.* [Online]
Available at: https://concurrent-tasks.js.org/
[Accessed 12 2109].

Shiotsu, Y., 2016. *Up Work.* [Online]
Available at: https://www.upwork.com/hiring/development/java-vs-javascript/
[Accessed 12 2019].

Singh, V., 2019. *Hackr.io.* [Online]
Available at: https://hackr.io/blog/java-vs-javascript
[Accessed 12 2019].

TK, 2018. *Free Code Camp.* [Online]
Available at: https://www.freecodecamp.org/news/functional-programming-principles-in-javascript-1b8fc6c3563f/
[Accessed 12 2019].

Uni, S., 2019. *Web.Standford.edu.* [Online]
Available at: https://web.stanford.edu/class/cs98si/slides/overview.html
[Accessed 12 2019].

Wiki, 2019. *Wikipedia, Pure Functions.* [Online]
Available at: https://en.wikipedia.org/wiki/Pure_function#I/O_in_pure_functions
[Accessed 12 2019].

Yellavula, N., 2017. *Medium.* [Online]
Available at: https://medium.com/dev-bits/everything-i-know-about-writing-modular-javascript-

applications-37c125d8eddf
[Accessed 12 2019].

CI346 - Programming Languages, Concurrency & Client Server Computing (Assignment 1 - John Vos)

ORIGINALITY REPORT

12%	4%	2%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Brighton Student Paper	4%
2	oberon2005.oberoncore.ru Internet Source	2%
3	Submitted to Colorado Technical University Online Student Paper	1%
4	Submitted to Ahlia University Student Paper	1%
5	www.itconference.co.kr Internet Source	1%
6	www.edureka.co Internet Source	1%
7	Submitted to Laureate Higher Education Group Student Paper	1%
8	Submitted to City of Glasgow College Student Paper	<1%

9	Ioannis Kostaras, Constantin Drabo, Josh Juneau, Sven Reimers, Mario Schröder, Geertjan Wielenga. "Chapter 7 Mastering the Core Platform", Springer Science and Business Media LLC, 2020 Publication	<1%
10	Submitted to School-Online Student Paper	<1%
11	sgwr2.student.eda.kent.ac.uk Internet Source	<1%
12	Submitted to NCC Education Student Paper	<1%
13	Submitted to Fareham College Student Paper	<1%
14	Submitted to City of Bath College, Avon Student Paper	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		