CI646 - Programming Languages, Concurrency and Client-Server Computing Report

By Jake S. T. Ward

Introduction

Java v JavaScript: Overview

Within this report, I will compare Java and JavaScript in terms of client-server computing, concurrency and security. I've chosen to compare these two languages because Java and JavaScript differ in many different aspects including but not limited to the subject of the report and CI646 module. Additionally, I will be showcasing other differences such as file system access, networking and the language's approaches to web app development.

Java and JavaScript were chosen due to their prominent use in the modern computing industry. Their rise to the level of industry standards is most likely due to their wide applications in many areas of both IT and computer science. Despite their similar naming schemes, both languages are from different families and backgrounds, with their own unique uses, use cases and features. It is because of the lack of overlap that means that both languages can be learnt completely independently of one another.

History and Development

The origins of Java and JavaScript can be found through very simple research as they are highly documented and publicized events. Both Java and JavaScript were developed and released over the course of the early to mid-90s. Java was created by James Gosling as part of a project at Sun Microsystems Inc in the mid-90s and saw its first official release in 1995, while JavaScript also found itself being invented in 1995 by Brendan Eich at Netscape. The Java project's team officially started work on the development of Java in 1991 with the initial goal of developing new advanced software for both embedded systems and network devices. Java was then quickly adopted by many developers and became a popular language for use in building enterprise applications due to its robustness, scalability, and high-tech security features, but additionally became synonymous with web development due to its ability to provide interactivity and multimedia. While the origins of Java and JavaScript were around the same time, their inventions were for completely different purposes. While Java was designed for embedded systems and network devices, JavaScript was designed to add interactivity to web pages. JavaScript was developed in 1995 as a scripting language for this very purpose. It was initially developed by Brendan Eich for Netscape 2 until it was handed to ECMA where it became their standard in 1997 and where they continued to develop it for the Mozilla Firefox browser. The shared naming scheme was due to Netscape’s support of Java applets within its browser. Despite their differences in both languages' original purposes, both languages have continued to evolve and expand in popularity over the years, with potential applications reaching far beyond what they were initially designed for.

Purpose

Today the uses for the two languages also differ substantially. Modern Java is designed to be platform-independent, meaning it doesn’t depend on any one operating system or hardware, making it robust and flexible in terms of what it can be used on. While JavaScript was designed to be for client-side web development, with the addition of frameworks like Node.js and React, JavaScript can be incredibly

flexible in terms of application as it can now be used for server-side functionality and much more. JavaScript without aid can is designed to be used by and is supported by all modern web browsers (like Safari, Edge, Firefox and Chrome) making it flexible in terms of the users' browser choice as well.

Java is often reserved for use in the building of server-side, client-side desktop and Android applications with IDEs available for development in all of these areas. JavaScript is advertised at the base as being a client-side scripting language allowing for the creation of dynamically updating content, animation, controlling multimedia and many other complex elements of modern web applications. While JavaScript is supported by browsers on mobile devices it does not have any mobile-exclusive applications like Java with android app development. However, mobile hybrid applications are starting to be written in JavaScript which, similar to the functionality of Java on Android, can access hardware and sensors found on mobile devices such as touch screens, accelerometers, gyroscopes and many other pieces of mobile-exclusive hardware.

The two languages also share many features in common such as:

- Both Java and JavaScript support Object Oriented programming despite Java being a language and JavaScript being a script
- Both support exception handling
- Both can/are supported and can be run in browsers
- Both can run on servers despite JavaScript requiring a framework
- Both make use of libraries to expand what the language is capable of and speed up development time
- Both make can use of Frameworks to expand what the language is capable of and speed up development time
- Both support garbage collection to delete code that is no longer in use. Although JavaScript deletes objects that have become unreachable while Java deletes code that is no longer in use. However, this still means that both languages use similar methods to free up memory.
- Both support Metaprogramming
- Both can be used with generics

But for this report, I am going to focus on the differences between the two.

Technical comparison

Compiled V Interpreted Languages

When researching the differences between Java and JavaScript the most frequently mentioned difference is the fact that Java is a compiled language while JavaScript is an interpreted language.

This difference matters little to the average person but developers will be massively impacted by this when they are choosing between the two languages. The reason it is impactful is because Java being a compiled language means that it checks the whole code all at once while it optimizes and converts the code into an executable file that will be read by the Java Virtual Machine (JVM) at a later date. Whereas JavaScript being an interpreted language means that the code is executed line by line as and when the lines are encountered, this means that errors can only be caught at the time of execution. JavaScript specifically is interpreted directly by the browser it has been launched by.

Both compiled and interpreted languages have benefits and drawbacks. The majority of syntax errors are caught during the compilation process for compiled languages like Java, which can save developers time and effort. Also, when compared to interpreted languages like JavaScript, the compiler's optimized executable code typically yields better performance. However, the compilation stage might make the development process more complex and may call for extra equipment and knowledge.

All the while interpreted languages like JavaScript provide developers with more flexibility and ease of use. Debugging is simplified by the ability to run code line by line, which allows problems to be detected and located in real-time. Moreover, JavaScript can be run directly within a browser without the need for an additional runtime environment like the JVM. The ability to interpret the code at runtime might introduce overhead that may negatively influence the user experience, and interpreted languages often operate more slowly than compiled languages.

Additionally, because Java has to be compiled before it is run, any structural or syntax errors will be caught before the program can even be run. While JavaScript will not catch any errors until the code is run. As well as this some popular Java IDEs can allow the developers to attach to the JVM to debug the code in real time. This means that while JavaScript only supports run-time debugging, Java can be defined as supporting Two-stage debugging.

Statically V Dynamically Typed

Another prominent difference between Java and JavaScript is that while Java is statically typed, JavaScript is dynamically typed. The reason this difference will impact developers is because statically typed languages like Java do all checking at compile time whereas dynamically typed languages like JavaScript do all their checking at run-time.

When comparing statically and dynamically typed languages there are many factors to weigh against each other, one of these factors is flexibility. In terms of flexibility dynamically typed languages come out on top. JavaScript as a dynamically typed language offers flexibility through the ability to have datatypes of variables be changed at runtime which is not a luxury Java affords. This streamlines development and makes writing code a lot easier for both newcomers and veterans of the programming industry. This ability also allows for fast prototyping which can dramatically help early stages of development when requirements may still be changing or being decided upon. While this ability makes errors less likely, the fact that there is no compilation and thus no type checking means that when errors do occur, they will be harder to find.

Another factor that can be used to compare statically typed languages and dynamically typed languages is robustness. Java is more robust than JavaScript due to its type-checking and error-catching capabilities that are only present in statically typed languages. The previously mentioned type checking upon compilation is a staple of statically typed languages which can prevent many errors and thus increase the reliability of code produced dramatically. The ability to have variables be able to change datatypes can actually hurt JavaScript's robustness as static typing's restriction to one datatype per variable can make code easier to maintain and read. This is because all datatypes will be forced to be used consistently across all code. Static typing can, however, inherently add rigidity and overhead, making the testing and prototyping of code a much harder task.

Class V Prototype Based

There are many differences between class-based languages like Java and prototype-based languages like JavaScript, however the biggest difference by far is down to how they handle inheritance. As languages supporting Object Oriented programming both Java and JavaScript support inheritance, however, Java as a class-based language must define a type to be instantiated upon runtime while JavaScript as a prototype-based language uses prototypes which are themselves already instances.

Object Orientated Programming

While Object Oriented Programming (OOP) is supported by both languages there are different ways in which the practice is carried out between programs. Seeing as how both languages support OOP it means that by extension, they both support the various aspects of OOP like polymorphism, encapsulation, and inheritance.

Java very strictly adheres to the OOP paradigm by nature of being class-based and strongly typed. Java ensures scalability and maintainability in code by encouraging the use of design patterns. Java uses classes to define its objects and is where methods and variables are to be organized. Strongly typed languages like Java require all variables to be declared with a specific type which is then enforced throughout the entire reach of said variable.

In contrast to Java's approach to OPP JavaScript is a weakly typed/untyped language meaning that the variables can be declared without a type and the language will attempt to parse any values into what it assumes is the appropriate data type, vastly increasing flexibility. This paired with the fact that the language is prototype-based allows for OOP to be used with far less restrictions over object creation and manipulation while still supporting the basic features of OOP like encapsulation and inheritance.

Functional Programming

Functional programming is yet another programming paradigm which both Java and JavaScript follow, however, the two go about its implementation in different ways. In JavaScript, Functions are treated like objects and can be passed into other functions as parameters as well as carry member variables. This flexible approach allows JavaScript to hold some programming solutions that are unique to the language. Alternatively, Java uses lambdas (released in Java 1.8) to adhere to functional programming, which is a way to filter and manipulate collections of data. Lambdas can only manipulate data collections and do not have much power beyond that. Overall Java supports functional programming through the support of lambda expressions and streams, with lambda expressions enabling developers to pass pieces of code as arguments into a constructor or method and streams being used to allow developers to process data functionally by changing together operations.

While Java supports the paradigm to a minimal extent JavaScript has been significantly impacted by it, so much so that it is thought to be one of JavaScript's core paradigms. It provides the ability to make first-class functions which means that functions can be assigned to variables, passed as arguments and returned as values from other functions ensuring that functions can be used the same as any data, making code easy to write in a way that is both modular and reusable.

Client-Server Computing

One of the main focuses of the CI646 module and areas in which Java and JavaScript can be compared and contrasted is client-server computing. Client-server computing refers to the architectural model in

which often a singular centralized computer acts as a server performing functions and responding to requests from other computers known as clients. Java is often used as a server-side language, while running on a server Java is most often used for generating and sending HTML code to client's browsers. From there the client will only interact with the resulting HTML and additional website-related elements, being oblivious to the functionality housed on the server and the heavy lifting that it performs.

On the other hand, JavaScript is a scripting language that was specifically designed for client-side functionality and usually within the client's browser of choice. JavaScript is still capable of running as server-side code but only with the addition of frameworks and/or plugins like Node.js and Express. As a client-dedicated language, it is usually responsible for handling all UI interactions, making AJAX calls, Fetch requests and other forms of sending requests to a server as well as receiving the resulting data. JavaScript's inclusion in web interfaces on the client side ensures that user interfaces are both responsive and dynamic, however, dynamically generated pages can also be generated on the server side and then sent to the client.

The level of interactivity offered by the two languages is another way that we can contrast within the area of client-server computing. Java tends to provide more interactivity on the server side by enabling real-time updates to be pushed to clients without requiring refreshing on the part of the user via the use of technologies like JSF and Java Servlets. JavaScript instead provides interactivity on the client side through dynamic and responsive user interfaces with animations and functions tied to different types of interactions like clicking, hovering and entering text. While all of this can be provided by base JavaScript, it can be implemented easier and faster through frameworks such as Angular and React which specialize in the creation of complex and modern web interfaces.

Looking at more ways to compare the two languages within the area of client-server computing we can focus on platform independence. Java is well known for its platform independence as it can run on any platform that has a JVM installed, making it very simple to write cross-platform apps, able to run no matter the hardware or operating system. JavaScript is tightly coupled to web browsers and the features that come with them. This means that platform independence is limited by machines capable of running browsers. With Node.js JavaScript can be run independently of a browser making it significantly less limiting.

There is another major difference between Java and JavaScript in client-server computing, and that is in their approach to security. Java has a long-established security model, which is based on a sandbox model. It restricts the actions that a Java program can perform to prevent it from accessing sensitive areas and/or resources on the system. This security model has been refined over many years and has become trusted in enterprise environments. On the contrary, JavaScript has had a more awkward and difficult security history. This is partly because of its tight connection to web browsers (due to the origins of the language), which have frequently been the subject of many attacks over the years.

To give specifics I've chosen a framework from Java and an embedded function from JavaScript to show how both languages send their client-server requests. For Java, one of the best ways to send client-server requests is through the spring framework (specifically using the Web module). The spring framework is an open-source application framework for Java that allows client-server requests to be sent via the restTemplate client or the new WebClient API. WebClient was designed to replace restTemplate and can submit synchronous and asynchronous HTTP requests. The spring web module as

a whole can be used to build RESTful services like APIs and handle many types of requests and responses with exception handling. It should be noted that Java also supports embedded functions for sending these requests like HttpURLConnection but Spring is a particularly popular method.

Figure 1: GET & PUT methods in Java with Spring WebClient

JavaScript's most modern solution to sending these requests is with the use of the Fetch API. This method is a browser API and the modern replacement for AJAX/jQuery requests, despite them still being supported, fetch is seen as superior as it is built off of promise objects, which has the advantage of simplifying the code greatly (especially in the context of using async/await). Fetch can be considered a more elegant solution than the spring web client as it can send all HTTP request methods without any need for the installation of anything besides base JavaScript. Requests in this format are simple to write and can even be generated simply with popular third-party apps like Postman. You can refer to the above and below figure to see the examples of how to write the most common requests (WebClient and Fetch API) with both methods.

Figure 2: GET & PUT methods in JavaScript with Fetch API

When comparing the two client-server request-handling methods we find that while they are both classified as libraries, WebClient cannot be used outside of its framework. Although Fetch is also dependent on the browser's support, almost all modern browsers support the Fetch API. Springs web client is designed to work with the spring web stack while Fetch is designed to work with all modern

browsers. Both are used for the purpose of creating and sending HTTP requests such as DELETE, PUT, GET and POST requests. Advantages of WebClient in Java include its ease of use, high configurability and compatibility with the rest of the Spring reactive web stack. It also has the advantages of supporting streams, non-blocking I/O, WebSocket requests and OAuth2 authentication. Its disadvantages include the method's dependence on the Spring reactive web stack and the fact that the method is exclusive to the Java language and so would be difficult to migrate code between languages. JavaScript on the other hand has the advantage of not requiring additional libraries, dependencies or frameworks. As well as its lack of dependencies, it is also easy-to-use and supports promises making it easier to work with async/await code. It also supports cookies, custom request methods and headers, giving greater flexibility for what is possible with simple HTTP requests. The disadvantages are that it cannot be run server-side due to the fact it is a browser-API plus it has no support of reactive streams and requires a lot of configuration or handling when it comes to CORS requests.

Unlike JavaScript which does not have any support for low-level network communication, Java can make use of the Java Socket API for that purpose. Socket is a Java class that can be called to provide two-way processes with a way to communicate over a network. It is a simplistic method for establishing a connection between client and server to allow for the two to exchange data.

WebSockets is a computer communication protocol allowing for real-time and bi-directional communication between a browser and server. Because all communication is carried over a long living connection all data can be sent without the need for more than one HTTP request to be made. Due to the fact that this method allows for real-time updates it is best suited for applications such as chat-based apps and online multiplayer games. While support is offered for this method in both Java and JavaScript, JavaScript has the WebSocket API provided by the browser on the client-side while Java commonly establishes these connections on the server-side using the Java API for WebSocket (Clients cannot connect to that long-form connection without using a WebSocket supporting library or framework like the SockJS library for both Java and JavaScript with Node.js or the Spring framework).

Security

Security is a critical concern in modern software development, and both Java and JavaScript have their strengths and weaknesses in this area. Java has a strong security model that relies on a sandbox environment to prevent untrusted code from accessing system resources. Additionally, Java has a robust authentication and authorization mechanism that ensures only authorized users can access sensitive information. However, this security model can come at a cost in terms of performance, and the sandboxing approach can limit the functionality of some applications. JavaScript, on the other hand, has weaker security features, and its sandbox environment is not as secure as Java's. JavaScript's dynamic nature also makes it easier for attackers to exploit vulnerabilities in code. However, to mitigate any potential security risks, JavaScript can be used in conjunction with security measures such as Content Security Policy (CSP). This approach involves defining a set of rules that specify which resources a web page is allowed to load, reducing the risk of attacks such as XSS and CSRF. In terms of security, both Java and JavaScript have their advantages and disadvantages, thus it is crucial for developers to carefully analyse the consequences of their security's design choices.

Another important aspect of security in client-server computing is the ability to authenticate and authorize users. Both Java and JavaScript have mechanisms for user authentication and authorization, but their approaches are different. In Java, developers can use the Java Authentication and

Authorization Service (JAAS) to provide authentication and authorization. JAAS allows developers to plug in different authentication and authorization providers, making it flexible and customizable. In JavaScript, authentication and authorization are typically handled on the server-side using frameworks like Passport.js. These frameworks provide a range of authentication strategies, including local authentication (username/password), social authentication (e.g., using Google or Facebook credentials), and two-factor authentication. While JavaScript's server-side authentication strategies are powerful and flexible, they require additional setup and configuration compared to Java's built-in authentication mechanisms.

An important security consideration in client-server computing is protection against attacks like the previously mentioned cross-site scripting (XSS) and SQL injection. Both Java and JavaScript have measures to mitigate these types of attacks, but again, their approaches are different. In Java, developers can use libraries like ESAPI (Enterprise Security API) or OWASP (Open Web Application Security Project) to prevent XSS and SQL injection attacks. These libraries provide encoding and validation functions that developers can use to ensure user input is safe. In JavaScript developers can use frameworks like Angular or React that have built-in support for XSS and SQL injection prevention. Additionally, JavaScript has measures like Content Security Policy (CSP) that can prevent attacks by blocking potentially dangerous scripts and resources from being executed. Overall, while both Java and JavaScript have measures to protect against attacks, the specific tools and techniques used may differ depending on the language and framework being used.

Going back to the topic of the WebSocket API, there are many relevant security threats that arise from its use considering how both Java and JavaScript can support it. A Security risk exclusive to WebSocket is Cross-site WebSocket hijacking (or CSWSH), this is when an attacker can establish a WebSocket connection with a user’s browser and can send data to or from the server while bypassing the Same Origin Policy. The most efficient way to prevent this is for the developers to validate the origins of the connection. WebSockets are also vulnerable to injection attacks such as XSS or SQL injection which can only be stopped through user validation and input sanitization. As well as being vulnerable to DoS attacks and being at risk of holding poor authentication depending on the design. Overall, the best practice must be followed to ensure that WebSocket is used securely in both languages.

Concurrency

Concurrency is the ability of a system to handle multiple tasks simultaneously and is a huge advantage for languages that support it, as it increases flexibility in terms of coding solutions and can be used to avoid code/UIs becoming unresponsive. Both Java and JavaScript support concurrency, but their approaches are different. Java has built-in support for multi-threading, allowing developers to create multiple threads of execution to handle complex tasks concurrently. This feature can be especially useful for server-side applications that need to handle multiple requests simultaneously. However, a disadvantage is that it also requires careful management of shared resources to avoid race conditions and other concurrency-related bugs. JavaScript, on the other hand, uses an event-driven model that relies on callbacks and promises to handle concurrent tasks. While this approach has the advantage of being more lightweight and easier to implement, the disadvantage is that it may not be as powerful as Java's multi-threading capabilities. That being said, JavaScript has been making strides in the area of concurrency with the introduction of new features such as Web Workers and Service Workers, which allow for the execution of scripts in separate threads. These features can offer similar levels of

concurrency as Java's multi-threading capabilities, but they are not yet as mature and may require more complex programming to implement effectively.

While being similar to concurrency, parallelism (or “parallel computing”) is a separate feature of Java and JavaScript. While concurrency refers to multiple tasks being carried out in an overlapping time period, parallelism refers to having a task broken down into smaller tasks and being executed over multiple resources such as cores. In Java, it can be achieved through the concurrent and stream packages to create threads and asynchronous tasks to be executed in parallel within a parallel stream API. While JavaScript relies on modern asynchronous programming techniques like the previously mentioned web workers and promises to execute parallel computing.

Another important aspect to consider when comparing concurrency in Java and JavaScript is how they handle synchronization. In Java, synchronization is achieved through the use of locks and monitors, which allow developers to control access to shared resources and prevent race conditions. In contrast, JavaScript relies on a single-threaded event loop that processes events in a queue. This means that JavaScript does not require explicit synchronization mechanisms, but it also means that long-running tasks can block the event loop, leading to a slower application. Although JavaScript has added new features like async/await to mitigate the problem, it is still crucial for developers to properly plan their applications to ensure efficient concurrency and avoid blocking the event loop.

The effect concurrency has on performance is another crucial factor to take into account. Java's multi-threading capabilities can improve performance by allowing multiple threads to execute simultaneously, but it can also lead to overhead and increased memory usage. In contrast, JavaScript's event-driven model is generally more lightweight and can be more efficient for certain types of applications. However, it can also suffer from performance issues when dealing with complex and long-running tasks. Ultimately, the choice between Java and JavaScript for concurrency will depend on the specific requirements of the application and the expertise of the development team in managing concurrency-related challenges.

On the topic of performance, a good example of how threads are used for concurrency is in the architecture of JavaFX apps. JavaFX, due to its specialization in the production of complex UI elements, often must deal with long-running code that would freeze the UI without the support of concurrency or parallelism. So, developers use threads to run long-running tasks in the background, so the main thread is free to handle event listeners for the UI. JavaScript by nature of being used for the interactivity of web pages needs to combat the same problem and so uses Web Workers to run long-running tasks in the background. Threads are used in many UI frameworks to maintain responsiveness as single-threaded programs, languages or frameworks will freeze if that single thread is overwhelmed. To best illustrate the concept, see the below pseudocode in the style of JavaFX.

```
// Create new thread to perform calculation in background
Thread thread = new Thread(() -> {
        // Perform longform calculation that WOULD freeze UI
        Result result = performLongCalculation();
        // Update the UI with the results
```

```
        Platform.runLater(() -> {
                ui.displayResults(result);
        });
});
// Run thread
thread.start();
```

File System Access

File system access is another area where Java and JavaScript differ. Java provides robust file system access capabilities, allowing developers to perform a wide range of file-related operations. JavaScript, on the other hand, has limited file system access capabilities due to security concerns. It can only access files within its own sandbox environment, and only if the user grants permission.

Networking

Networking is another area where Java and JavaScript differ. Java provides comprehensive networking APIs and supports a wide variety of network protocols which is what allows developers to build robust networking applications. JavaScript, on the other hand, relies on the browser's networking capabilities and can only communicate with servers via HTTP requests.

It is Java's support of network protocols such as TCP and SSL that makes Java well suited to the development of any application which require more than basic HTTP requests. But while JavaScript can be inherently limited to HTTP requests by nature of the browser, libraries like Axios and Request exist to supplement the browser with the additional networking capabilities required. However, it should be noted that while JavaScript can be supplemented with network functionality beyond HTTP, these libraries still pale in comparison to the power or comprehensiveness of Java's provided API's.

The fact that the networking capabilities of Java reach to low level like TCP, is advantageous because it allows developers using Java to create fully customized networking applications, which can be helpful when control over traffic is desired like how it is in real-time processing applications or high-performance computing. Juxtaposing this, JavaScript's reliance on the browser for networking capabilities is a disadvantage as it can severely limit the control developers have over networks. However, new technologies like WebSockets and others have been developed to allow JavaScript to handle real-time communication as well as other advanced networking tasks, allowing it to have a comparative level of control over a network to Java.

Web Apps

Both Java and JavaScript are popular choices for building web applications. While JavaScript is designed to be used for client-side web development, Java is frequently employed to create server-side web applications. But now since the advent of Node.js, JavaScript can also be used to create server-side web apps. Node's popularity and conventionality are evidenced by the surpassing of one billion downloads in 2018.

When it comes to building web applications, Java and JavaScript offer different advantages and disadvantages. It is Java's scalability and reliability that makes it the perfect choice for large-scale enterprise applications. Additionally, the Java ecosystem provides such a wide range of libraries and frameworks that can simplify the development process, that it has been adopted as one of the de facto languages for android app development. However, the main drawback of Javas usage is that it can take a long time to develop in and uses a lot of memory and resources when running. JavaScript, on the other hand, is faster to develop in and more lightweight, making it ideal for small to medium-sized applications. With the popularity of Node.js, JavaScript has also become a viable option for building scalable server-side applications. However, the JavaScript ecosystem can be fragmented and less mature compared to Java, and it may require more manual configuration to get things up and running.

Both Java and JavaScript offer a wide range of frameworks and libraries to help developers build web applications more efficiently. For Java, popular web frameworks include Spring, Struts, and JavaServer Faces, each with its own set of advantages and disadvantages. While JavaScript has been provided with a number of well-known frameworks, like Angular, React, and Vue.js. These frameworks have been increasingly popular in recent years due to their simplicity and ease-of-use speeding up both development time and learning curves. While Java's web frameworks tend to be more robust and feature-rich, JavaScript frameworks excel in their ability to build modern, dynamic user interfaces that provide a rich user experience. Ultimately, the choice between Java and JavaScript for web development will depend on the specific needs of a given project and the preferences of the development team, as neither is definitively better than the other.

IDEs

Integrated Development Environments (IDEs) are software applications that facilitate the development of software programs by providing a comprehensive/user-friendly environment for writing, testing, and debugging code. Java and JavaScript developers have access to a range of IDEs, each with unique features and capabilities that cater to their specific needs.

Eclipse, an open-source IDE with support for numerous programming languages, including Java, is one of the most well-known IDEs for Java development. Eclipse includes a range of features that make it well-suited for enterprise application development, such as support for multiple project types and integration with popular build tools like Maven and Gradle. Additionally, Eclipse provides a robust debugging environment and supports code refactoring, which can make code maintenance easier.

In contrast, JavaScript developers often use lightweight text editors like Visual Studio Code or Sublime Text instead of traditional IDEs. These editors provide a range of features, such as syntax highlighting, code completion, and debugging tools, while being faster and more lightweight than traditional IDEs. They can also be easily extended with plugins and packages that add functionality specific to JavaScript development, such as support for popular JavaScript frameworks like React and Angular.

Test-Driven Development

Test-Driven Development (TDD) is a method of implementing software that involves creating unit tests (which will automatically be executed upon running the code) before implementing the actual software that the unit tests will be testing. By interviewing coding, testing and design through the use of refracturing, defect rates go down, overheads are offset and many report that it helps improve design

and overall technical quality. TDD is relevant as it can be applied to both Java and JavaScript, but their implementations can differ.

When applying TDD with any programming language the first concern should be setting up unit tests. This is the first juxtaposition between the languages, as Java and JavaScript, while both requiring frameworks to step up unit testing, require different frameworks. Two examples of the most popular frameworks for Java unit testing include JUnit and TestNG while JavaScript makes use of frameworks like Jest, Mocha and Jasmine. The aforementioned Java frameworks provide a robust set of tools for writing and running tests and can be easily integrated into popular IDEs like Eclipse or the more modern IntelliJ. While the JavaScript frameworks for unit testing work in a similar fashion, JavaScript's dynamic typing system can make writing comprehensive tests challenging due to the fact that it becomes harder to ensure all scenarios are covered. Conversely, Java by nature of being statically typed, makes it easy to write tests to cover a wide range of scenarios, seeing as how the compiler will catch a majority of errors before the code is even executed. Additionally, JavaScript can be held back by its asynchronous programming in TDD as the asynchronous programming model can make it hard to write tests covering all execution paths.

Generic Types and Metaprogramming

There are variations between how Java and JavaScript approach generic types although both can make use of the style of computer programming. Generic programming is a popular programming method which entails algorithms being written in terms of types, that are defined later and then instantiated when required for a specific type (that is provided by the input parameters). The use of generics allows classes and interfaces to be used as parameters when defining other classes, interfaces or methods. While Java has in-built support for generic types, JavaScript does not. But generics can still be simulated in JavaScript by using prototypes and inheritance. Java enforces all generics on compilation and type erasure is used to prevent type information from being available at run-time, as no new classes can be created for parameterized types whilst type erasure has been implemented.

Metaprogramming is defined as a programming technique that allows programs to use other programs as their input data. Languages supporting metaprogramming can in essence design, read, generate and alter other programs and well as alter itself during runtime. Metaprogramming is supported in Java through reflection, as it allows objects to be inspected and modified during runtime. Java annotations are a large part of how this can function with Java, as the annotation provides a type of metadata about the program which isn't a part of the program itself. These annotations can be read by other programs or libraries after compilation allowing for metaprogramming to be possible. JavaScript can support metaprogramming but it is achieved using the in-built eval function to execute dynamically generated code and through the use of proxy objects which lets object be modified at runtime similar to Java's reflection. JavaScript does not use annotations but uses decorators for the same purpose. By using decorators or annotations the same program can be run with different behaviour as it changes it’s behaviour, depending on the input from another program (common examples of this can be found in IDEs and Frameworks).

Conclusion

In conclusion, Java and JavaScript share similarities and display differences in many areas including Concurrency, Client-Server computing and Security. While these differences and similarities are visible

within the languages alone, they can also be seen within frameworks and libraries linking to the languages as well as the APIs utilized by them. Many of the disadvantages of the languages can be supplemented by other products like frameworks to make up the difference between the languages. Overall, both Java and JavaScript have their own unique uses and should be chosen on a case-by-case basis for projects and neither is objectively better than the other.

Over the course of the project, I've learned about their differences in terms of metaprogramming, generic types, TDD, IDEs, Networking, File System Access, Concurrency with threads, Security, WebSocket, Client-server computing, the origins of the languages and more basic architectural differences like statically typed and dynamically typed or compiled and interpreted languages. I explored each of these interweaving subjects in the report with examples and smaller details often not mentioned in comparison pieces on the two languages. I believe that I've outlined the information in a clear, concise, yet in-depth and technical way to display understanding of both the technical comparison and the areas of focus, being client-server computing, security and concurrency.

Estimated grading

Technical comparison: A
Focus on client-server computing, security, concurrency: A

References

Åkesson, T. and Horntvedt, R., 2019. Java, Python and Javascript, a comparison.

Fink, S., Knobe, K. and Sarkar, V., 2000. Unified analysis of array and object references in strongly typed languages. In *Static Analysis: 7th International Symposium, SAS 2000, Santa Barbara, CA, USA, June 29-July 1, 2000. Proceedings 7* (pp. 155-174). Springer Berlin Heidelberg.

Tratt, L., 2009. Dynamically typed languages. *Advances in Computers*, *77*, pp.149-184.

de Oliveira Guimaraes, J., 1998. Reflection for statically typed languages. In *ECOOP'98—Object-Oriented Programming: 12th European Conference Brussels, Belgium, July 20–24, 1998 Proceedings 12* (pp. 440-461). Springer Berlin Heidelberg.

Koskela, L., 2007. *Test driven: practical tdd and acceptance tdd for java developers*. Simon and Schuster.

Dean, D., Felten, E.W. and Wallach, D.S., 1996, May. Java security: From HotJava to Netscape and beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy* (pp. 190-200). IEEE.

Drnasin, I., Grgić, M. and Gogić, G., 2017. JavaScript access to DICOM network and objects in web browser. *Journal of digital imaging*, *30*, pp.537-546.

What is the difference between Java and JavaScript?: Software guild (no date) *The Software Guild*. Available at: https://www.thesoftwareguild.com/faq/difference-between-java-and-javascript (Accessed: February 27, 2023).

Tilkov, S. and Vinoski, S., 2010. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), pp.80-83.

Steflik, D. and Sridharan, P., 2000. *advanced JAVA Networking*. Prentice Hall Professional.

Yu, D., Chander, A., Islam, N. and Serikov, I., 2007. JavaScript instrumentation for browser security. *ACM SIGPLAN Notices*, *42*(1), pp.237-249.

Lewandowski, S.M., 1998. Frameworks for component-based client/server computing. *ACM Computing Surveys (CSUR)*, *30*(1), pp.3-27.

Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Lea, D. and Holmes, D., 2006. *Java concurrency in practice*. Pearson Education.

Togashi, N. and Klyuev, V., 2014, April. Concurrency in Go and Java: performance analysis. In *2014 4th IEEE international conference on information science and technology* (pp. 213-216). IEEE.

Wang, J., Dou, W., Gao, Y., Gao, C., Qin, F., Yin, K. and Wei, J., 2017, October. A comprehensive study on real world concurrency bugs in Node. js. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 520-531). IEEE.

Boduch, A., 2015. *JavaScript Concurrency*. Packt Publishing Ltd.

Kleiman, S., Shah, D. and Smaalders, B., 1996. *Programming with threads* (p. 48). Mountain View: Sun Soft Press.

(no date) *Spring Framework Documentation*. Available at: https://docs.spring.io/spring-framework/docs/current/reference/html/ (Accessed: March 13, 2023).

Fetch API - web apis: MDN (no date) *Web APIs* | *MDN*. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (Accessed: March 13, 2023).