

Justis Dutt

Dr. Megan Thomas

CSU Stanislaus CS4100

Fall 2024

A Comparison of Type Systems: Java vs. JavaScript

Abstract

This paper is a comparative analysis of the type systems of Java and JavaScript, discussing how static typing in Java and dynamic typing in JavaScript influences software development. Type systems in programming languages influence performance, error detections, debugging, and maintainability, making the choice between static and dynamic typing significant. Java's strong, statically typed system provides features like compile-time type checking, which increases reliability and reduces runtime errors. This is especially important for large applications. On the other hand, the dynamic, weakly typed system of JavaScript provides great flexibility and rapid prototyping, but also opens up the possibility for a large amount of runtime errors and debugging complications. This paper provides insights into the advantages and disadvantages of each type system, with the goal of assisting programmers in making informed choices about which type system best suits their needs.

1. Introduction

Type systems play a vital role in programming languages by shaping how languages handle variables, expressions, and interactions within code. Type systems playing a vital role is shown in two of the most popular programming languages in the world, Java and JavaScript. The two programming languages vary greatly with regard to their type systems. Java is a static, strongly-typed language that requires explicit type definitions for variables and expressions during compile-time [11]. In contrast, JavaScript is a dynamic and weakly typed system where variables can change dynamically during runtime. JavaScript's type system can be flexible but opens different risks that Java's type system prevents [8, 11]. By comparing two contrasting type systems in Java and JavaScript, this paper aims to provide a clearer understanding of the advantages and disadvantages associated with static and dynamic typing, to help programmers have better insights into the suitability of each language in different programming contexts.

The importance of understanding type systems goes beyond each type system's immediate advantages or disadvantages. Type systems fundamentally affect the process of

development, impacting key components like performance optimization, error detection, debugging, and maintainability. Additionally, the long-term maintainability of software often depends on how well type-related issues are managed throughout the project lifecycle, making the choice between static and dynamic typing one that affects the life of a project [12].

This paper explores how these different type systems influence software development, particularly in areas such as performance, optimization, error detection, debugging, and maintainability. This analysis is especially relevant for programmers who are deciding between Java and JavaScript for new projects or those seeking to understand how static or dynamic type systems work.

2. Background

2.1 Overview of Type Systems

2.1.1 Definition (Type System): A *type system* is a set of rules that assigns a type (like an integer, string, or float) to various components of a program such as variables, expressions, and functions [3].

Type systems are often characterized by how strongly or weakly they enforce type constraints. A strongly-typed system, like Java, has strict rules about how different types can interact, preventing type-related errors. On the other hand, a weakly-typed system, like JavaScript, allows for implicit type conversions or type coercion which makes values of different types automatically convert to a different type as needed [4]. The differences in type strength and when type checking occurs (compile time vs, runtime) between Java and JavaScript, significantly impact how each of these languages is interacted with and perform.

2.2 Java's Static Type System

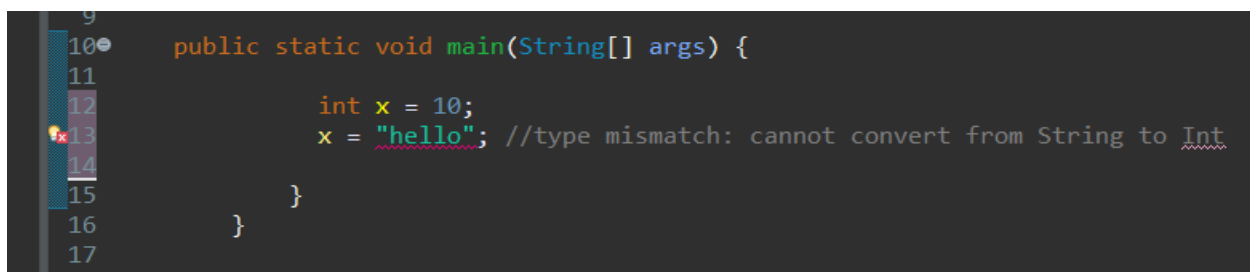
2.2.1 Definition (Static Typing): Every variable and every expression have a type known at compile time [2, 11].

2.2.2 Definition (Strongly-Typed Systems): Java is considered to be a *strongly-typed system*. In a *strongly-typed system*, the programmer cannot work around the restrictions imposed by the type system [11, 12].

The static type system in Java assigns every variable and expression a type that can be determined at compile-time [11]. By doing this, the statically typed language in Java can ensure that every variable and expression type is checked before the program is executed, allowing the compiler to detect type-related errors [11]. Type checking during compile-time provides a high

level of reliability and predictability by ensuring that the types of variables and expressions remain consistent throughout the entire program [11]. This can be a big advantage for statically typed languages over dynamically typed languages, where types may change during runtime, as it can prevent unexpected behaviors from happening when a type is unexpectedly changed [4, 5]

The Java programming language is strongly typed, which makes sure that a variable can only hold data that are compatible with its type, and controls what operations are valid for that type. For example, if you try to assign a string to an integer variable, a compile-time error would be thrown [see Figure 2.2.1]. The strong static type system in Java helps catch errors very early in development by reducing the ability for a type-related error to occur during runtime [2, 11].



```

9
10 public static void main(String[] args) {
11
12     int x = 10;
13     x = "hello"; //type mismatch: cannot convert from String to Int
14
15 }
16
17

```

Figure 2.2.1: MADE BY AUTHOR. Example of a type mismatch in Java due to a strong static type system.

Java has two main categories of types, primitive types and reference (non-primitive) types. In parallel, the data types that can be stored in variables, passed as arguments, returned by methods, and manipulated have two categories, primitive values and reference values [11]. Primitive types are the most basic data types: boolean, byte, short, int, long, char, float, and double [11]. These types are not objects and store their values directly in memory. This is designed for efficiency in arithmetic and logical operations [7].

Reference types include class types, interface types, and array types. There is also a special null type. These types do not hold the actual data but a reference object or array that is dynamically created during program execution [11]. Objects in the Java programming language are instances of classes, created using the *new* keyword, and inherit all methods of class Object, which is a superclass of all other classes [1]. Additionally, string literals in Java are represented as instances of the String class, which allows for manipulations and operations on text that are hard to handle [1, 11].

2.3 JavaScript's Dynamic Type System

2.3.1 Definition (Dynamic Typing): Variables are determined at runtime based on the variable's value at the time [13]. Thus, you do not need to specify the data type of a variable when you declare it. *Dynamic typing* in JavaScript automatically converts as needed as the script is run during execution [13].

2.3.2 Definition (Weakly-Typed Systems): JavaScript is considered to be a *weakly-typed language*, as it allows implicit type conversion. Operations may produce results by automatically converting one type to another without explicit data types being declared [13].

2.3.3 Definition (Type Coercion): *Type coercion* in JavaScript happens implicitly, which means JavaScript automatically converts one type to another when two different types are involved in the same operation [10]. This helps with flexibility but can cause unexpected behavior if managed poorly [10].

2.3.4 Definition (Just-In-Time Compilation): Code is compiled into machine code at runtime. This allows JavaScript engines to optimize code based on the types the program encounters during runtime, this allows for a much faster execution time compared to interpreted languages [10].

JavaScript's type system is dynamic and weakly typed [10]. This means that variables in JavaScript are not bound to any data type or their rules during compile-time. This is the flexibility that JavaScript developers love about the programming language, as it allows a programmer to assign different types of values to the same variable throughout a program's execution [8]. For example, a variable can start as a number, be reassigned as a string, and then be reassigned again as a boolean (see Figure below).

```
1 let variable = 42; // variable is now a number
2 variable = "Hello"; // variable is now a string
3 variable = true; // variable is now a boolean
```

Figure 2.3.1: MADE BY AUTHOR. The example shows the same variable being reassigned in JavaScript.

JavaScript being a weakly-typed language, allows for implicit type coercion between incompatible types [10]. This can be a feature and a pitfall. For instance:

```
1 Number = 12;  
2 Result = Number + "3"; //JavaScript Coverts Number into String for concatenation  
3 console.log(Result); // Output is 123
```

Figure 2.3.2: MADE BY AUTHOR. This example shows an integer being added to a string in JavaScript.

JavaScript coerces the variable `Number`, an integer, into a string to perform string concatenation. This results in the string “12” and the string “3” being concatenated, and the result is “123”. This flexibility can make JavaScript feel intuitive and enable faster prototyping [10, 13] since programmers do not need to worry very much about explicitly casting types. JavaScript allows programmers to focus their time on functionality by attempting to convert one type to another when an operation involves mismatched types rather than throwing a type error. This can simplify code and speed up prototyping but also introduces subtle bugs, as programmers may not be able to always anticipate when or how these conversions will occur.

JavaScript categorizes its values into primitive types (`Number`, `String`, `Boolean`, `Undefined`, `Null`, `Symbol` and `BigInt`) and reference types (objects). Primitive types in JavaScript are fixed and these types hold their values directly [4]. This means that once a primitive is created, it cannot be changed. For example, when you manipulate a string, operations such as concatenation create a new string rather than modifying the existing one in memory. This immutability allows for efficient memory storage by avoiding unintended side effects [10]. When a value is mutable, changing the value of one reference can change the value of another reference to the same data, which could cause unintended side effects. Since primitive values in JavaScript are immutable, no other part of the program can be affected when a variable is reassigned to a new value since a new primitive is created when modified.

While primitive types are immutable, objects in JavaScript are mutable and are stored as references, which allows for dynamic modification [8]. Objects in JavaScript are reference types and work like reference types in Java. This means that variables do not store the actual object data but a reference to the memory location. Unlike primitive types, objects like arrays or functions allow complex data organization and manipulation [8]. A programmer can have an object add to itself, remove from itself, or modify itself without having to create a new object to do so. For example, an array is able to change or modify itself by pushing a new item into the array or by changing the value of an object’s property. This flexibility in objects allows programmers in JavaScript to manipulate data structures dynamically, modifying data as the program is run.

3. Comparison of Type Systems

3.1 Performance and Optimization

Java and JavaScript both have distinct advantages they receive and challenges they are faced with due to their type systems. Java's static typing requires that types are declared and checked at compile-time. This allows the Java Virtual Machine (JVM) to optimize code before execution [10]. Being able to optimize code before execution helps the JVM use system resources efficiently and leverage hardware capabilities in the best possible manner, which makes performance and memory use patterns more predictable [7]. Specifically, type information being available at compile-time enables the JVM to optimizations such as memory allocation, garbage collection, and inline catching that improves efficiency at runtime [7].

By contrast, JavaScript is dynamically typed. The lack of static type constraints makes performance rely on Just-In-Time compilation (JIT). Most modern JavaScript engines use JIT compilation (see definition 2.3.4) which compiles code at runtime and lets the engine adaptively optimize code based on the actual usage patterns and types that it sees at runtime [10]. This adaptive approach lets JavaScript execute quickly by taking advantage of several JIT optimizations like hot code paths and deoptimizations of unused operations [10]. This means that JavaScript is able to be more efficient by prioritizing optimizations on frequently used code paths, hence the name “hot”, while deoptimizing less used paths and operations.

Figures 3.1.1 and 3.1.2 show a basic example contrasting variable assignment and type handling in Java and JavaScript. In Java, type declarations are enforced and make sure that variables “int a” and “int b” are optimized for integer operations during compilation, which makes it faster during execution time [4]. But in JavaScript, variables “a” and “b” can change their types freely, which can significantly slow down the operation because of runtime inference of types and potential coercion of types, as explained in section 2.3.

Java: Static Typing

```
int a = 5;
int b = 10;

int result = a + b; // Compile-time optimization for integer addition
```

Figure 3.1.1: MADE BY AUTHOR. Example shows type handling in Java.

JavaScript: Dynamic Typing

```
1 let a = 5;
2 let b = "10";
3 let result = a + b; // Type coercion results in "510" (string concatenation)
```

Figure 3.1.2: MADE BY AUTHOR. Example shows type handling in JavaScript.

In the Java example, $a + b$ will result in an integer addition, while in JavaScript, due to type coercion, $a + b$ results in “510”. This is because JavaScript allows for operations with mixed types, which can introduce performance costs like the example shown above. The JavaScript engine must solve these problems at runtime through type coercion since the type system is dynamic and weak. This shows how Java and its static type system can have better and more predictable performance but will lack the flexibility in performance that JavaScript’s dynamic typing has.

3.2 Error Detection and Debugging

With Java, many errors and bugs can be detected by the static type system at compile-time rather than run time. Strict static typing in Java constrains types to ensure a variable will not suddenly change types. It is this very type constraint that allows the compiler to detect things such as mismatched types, undeclared variables, and incompatible method calls at the beginning of the development cycle rather than later [2]. Strict static typing in Java constraining types helps with program stability as well as performance [2]. For example, figure 3.2.1 shows an attempt to pass a string where an int is required results in a compile-time error, which halts the code from being able to execute until the issue is resolved. Since Java is able to catch type-related errors during compilation, the likelihood of encountering type-related errors during execution is minimal [11]. This makes Java a preferred choice for many large applications where reliability is critical, such as medical devices [6].

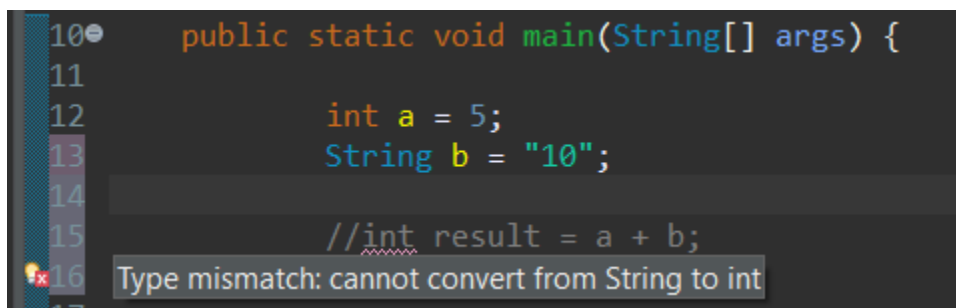


Figure 3.2.1: MADE BY AUTHOR. Example shows Java’s error message when there is a type error, in this case a type mismatch.

Java also has comprehensive error messages that assist programmers pinpoint the location and type of issue they are faced with. For example, a compile time error message might specify

the line number and the type of bug, making the process of debugging faster and easier [10]. Since Java has a strict static type system, Java will throw error messages whenever there is a type error. Figure 3.2.1 shows how Java's strict type constraints catch a type error during compilation, preventing runtime errors and allowing for a more predictable debugging process.

On the other side, the dynamic and weakly typed system of JavaScript brings flexibility but also makes the processes of error detection and debugging more difficult as it does not check the types of variables at compile-time [4]. This means that type-related errors are left to be dealt with at run time, where they are much harder to find and fix. This makes debugging a more difficult process because JavaScript can also perform implicit type conversions through a process called type coercion, as explained in section 2.3. For instance, JavaScript may coerce an integer into a string, leading to unexpected concatenations instead of an integer addition. While this is useful at runtime for rapid prototyping, it also makes code in JavaScript more error-prone, since the programmer will need to anticipate unexpected type coercions and handle them [4, 10].

Due to these characteristics, debugging in JavaScript tends to be more difficult for programmers when applications are at a large scale. This is because type related errors can spread through multiple layers of code without being caught. There are debugging tools and runtime checks, such as those offered by Node.js and browser developer tools [7], that provide some error management, but they do not catch all type related problems. Programmers also rely on additional tools, such as type-checking libraries, to actually introduce static typing features into JavaScript to help reduce type related errors at runtime [7].

Again, the difference in error detection between Java and JavaScript shows how the type systems of these two languages are different. Static typing in Java is structures and provides reliable development with high predictability of errors, while dynamic typing in JavaScript allows for flexibility that can introduce runtime errors and requires extra time and work during debugging.

3.3 Maintenance

Java's static type system and JavaScript's dynamic type system have huge impacts on long-term code maintainability [9]. Large projects will always grow and evolve, and as they do, they will integrate more components and third-party libraries. This means that maintenance will always be a main concern in large applications. Java and JavaScript both have advantages and disadvantages directly linked to their type systems.

Java's static type system does type checking at compile-time which helps keep maintainability consistent, as it enforces a certain structure throughout the entire program [9]. Since types must be explicitly declared, it reduces the chances of unintended behavior while changing or upkeeping code, as the compiler detects many potential bugs early on. For example, if a programmer attempts to pass an object of the wrong type to a method, then a compile-time error is thrown. This enforcement of type compatibility reduces the need for extensive runtime testing, making refactoring code and extending functionality easier since the type guarantees help maintain code integrity [9]. Java's static type system promotes predictable code and can be very useful in large applications where preventing unforeseen side effects critical [9].

JavaScript, being dynamically typed, brings great flexibility, making prototyping easier and speeding up development cycles in small projects. JavaScript's flexibility becomes a problem when applications grow in size and complexity, bringing along a lot of maintenance problems. Weak typing and implicit type coercion in JavaScript can lead to unintended behaviors of types that may make the debugging process longer and more tedious. JavaScript's type system increases the chances of runtime errors when data types shift unexpectedly [8]. This may cause some maintenance issues for programmers as they are forced to find type related errors on their own. For example, without enforcing type declarations, variables sometimes get reused for purposes other than originally defined, making code more challenging to read. This also makes it more likely for bugs to appear when the application grows in size [8].

4. Case Studies

Java has become a key programming language for enterprise applications due to its strong statically typed system that ensures type safety and allows for maintainability at a large scale. A good case is its use in banking systems, where reliability and predictability cannot be sacrificed. For example, a case study on the use of Java-based systems by a large financial institution found that runtime failures and debugging were reduced through Java's static type systems catching errors at compile-time [5]. By catching type-related errors at compile-time, programmers can ensure higher stability as well as minimize errors during (costly) production time [5].

In contrast, JavaScript's dynamic typing has taken over web development, particularly in UI and front-end applications [13]. A case study of a startup building a single-page application (SPA) showed that JavaScript's flexibility sped up the prototyping process by allowing programmers to iterate and implement features faster because of its weak dynamically typed

system. Due to the weak typing of JavaScript's type system, variables are allowed to be dynamically reused [13]. Allowing variables to be dynamically reused provides flexibility that programmers can take advantage of to speed up prototyping [13].

5. Conclusion

In comparing Java's static typing and JavaScript's dynamic typing, this paper reveals that each type system has distinct advantages and disadvantages, which shape software development in unique ways. The strong, statically typed system of Java brings a high degree of reliability since it can catch errors early during compile-time, making the development of large applications predictable and manageable [11]. This helps with performance optimizations and maintainability in the long run, which is vital in environments where stability is key, like medical equipment and enterprise applications [6, 13]. In contrast, JavaScript's dynamic, weakly typed system maintains flexibility that allows for quick prototyping and quicker development for web development [13]. This flexibility comes with an increased risk of runtime errors and a more tedious debugging process.

Whether Java or JavaScript should be a language of choice depends on the needs of the project and the concerns of the programmers. Applications that require high performance and strict control of errors are best suited for Java and its strong, statically typed system. Projects that place a high value on flexibility can see significant benefits from taking advantage of the dynamic type system of JavaScript. To select the programming language that best suits the goals of a project and optimizes efficiency and effectiveness, a programmer must understand the implications of each available type system.

Bibliography

- [1] Abadi, M., & Cardelli, L. (1996). *A Theory of Objects* . New York City : Springer Publishing Company.
- [2] Bloch, J. (2017). *Effective Java, 3rd Edition*. Indianapolis: Addison-Wesley Professional.
- [3] Contributors, W. (2024). *Type Systems*. Retrieved from Wikipedia:
https://en.wikipedia.org/wiki/Type_system
- [4] Flanagan, D. (2020). *JavaScript: The Definitive Guide, 7th Edition*. Sebastopol: O'Reilly Media, Incorporated.
- [5] Flanagan, D., Evans, B. J., & Clark, J. (2023). *Java in a Nutshell, 8th Edition*. Sebastopol: O'Reilly Media, Incorporated.
- [6] International, E. (2024, July). *Common Software Languages For Medical Devices* . Retrieved from Emma Interantional: <https://emmainternational.com/common-software-languages-for-medical-devices/>
- [7] Ismail, M., & Suh, G. E. (2018). Hardware-Software Co-optimization of MemoryManagement in Dynamic Languages. *Proceedings of the 2018 ACM SIGPLAN International Symposium on Memory Management*, 45-58.
doi:<https://doi.org/10.1145/3210563.3210566>
- [8] Kereki, F. (2023). *Mastering JavaScript Functional Programming - Third Edition*. Birmingham: Pakt Publishing. Retrieved from Oracle.
- [9] Loy, M., Niemeyer, P., & Leuck, D. (2023). *Learning Java, 6th Edition*. Sebastopol: O'Reilly Media, Incorporated.
- [10] Mozilla. (n.d). *JavaScript Guide*. (Mozilla Developer Network) Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types
- [11] Oracle. ((n.d)). *Java Standard Edition 7 Documentation*. Retrieved from Oracle:
<https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.3>
- [12] Ramsey, N. (2022). *What is the difference between a strongly typed language and a statically typed language?* Retrieved from stackoverflow:
<https://stackoverflow.com/questions/2690544/what-is-the-difference-between-a-strongly-typed-language-and-a-statically-typed>
- [13] Zakas, N. C. (2010). *High Performance JavaScript* . Sebastopol: O'Reilly Media, Incorporated.

