THE COMPARİSİON OF JAVA AND C PROGRAMMİNG LANGUAGE

In the last century, it would not be an exaggeration to claim that life is becoming more digitized at a fast pace. The fundamental reason is technological devices, which play a significant role in our daily lives. Simply put, everything from our computers and televisions to smartphones is a product of technology. This digitalization began with massive inventions like transistors, capacitors, and chips. Afterward, a new field emerged for controlling electronic devices, which we can refer to as programming or computer science and engineering. How can we communicate with electronic devices? The answer is software, which refers to a set of instructions and data that controls the operation of computer hardware to perform specific tasks or functions, encompassing applications, operating systems, and programming code. But what exactly are these 'instructions'? How do computers understand them? A 'set of instructions' is a programming language, which helps us communicate with computer hardware. In this discussion, we will focus on programming languages and their features, and delve into two prominent programming languages, Java and C programming language. Let’s investigate the introduction of programming languages and these two languages. Programming languages have evolved over time. They started with assembly languages linked to specific hardware. Higher-level languages, like FORTRAN for science, LISP for AI, COBOL for business, and ALGOL for algorithms, emerged. In the late 1960s, C was created, emphasizing step-by-step problem-solving, hardware control, and efficient compilation. It's essential for operating systems and more. In the mid-1990s, Java was born, with a focus on portability, efficient execution on different platforms, and security. It's crucial for web apps, mobile apps, and large-scale systems. After introduction of two languages and programing languages, We deeply step on some topics which are Language Evaluation Criteria, Implementation method, Syntax and Semantics, paradigm, efficiency, availability and learning curve.

In this section, we will explore the differences between the Java and C programming languages in terms of ***Language Evaluation Criteria*****. Readability** is one of the most crucial criteria for assessing a programming language because it ensures that programs are written and understood easily. One aspect of

readability is **Overall Simplicity.** Symbols in programming have meanings, much like mathematical operators (+ - * /). We expect these symbols to be used for mathematical assignments involving integers and floating-point numbers, such as:

int num1 = 5, num2 = 7;

num1 + num2;

Both languages we are comparing can execute this part of the code. However, if you want to concatenate two strings using the `+` operator, what should the programming language do logically? We expect it to combine the strings, like this:

String name = "Ziya Kadir", lastname = "TOKLUOGLU";

String full_name = name + lastname; // the result would be "Ziya Kadir TOKLUOGLU"

Java can execute this code, but C cannot. C can achieve the same result by using functions. Furthermore, we can consider operator overloading, which is not supported in Java and C. However, Java has the `toString` method. This allows any object in Java to be printed using a simple print function if the class has a `toString` method. These factors indicate that Java is simpler to work with compared to the C programming language**. The other sub content is Orthogonality.** Orthogonality in a programming language means that a relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language. Java is like building with LEGO bricks that fit together easily, as it was designed with a strong focus on making different parts work well together. You can combine features like inheritance and interfaces without problems, which gives Java a high level of orthogonality. In contrast, C is more like using a basic construction set, where you must be careful with how you combine parts. It offers flexibility but doesn't automatically ensure that all the pieces fit together seamlessly, which means it has a lower level of orthogonality. The other sub-topic is **Data types.** The presence of adequate facilities for defining data types and data structures in a language is

another significant aid to readability. For example, if there is no Boolean type in a language, we probably assign int 0 for wrong 1 for correct to variable. Like this:

int control_variable = **1;**

The meaning of this statement is not clear even if the variable name is logical. Instead of this we use Boolean type like:

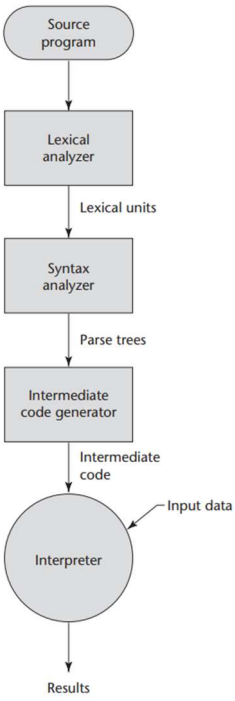
boolean control_variable = **true;**

that provides programming language is more understandable. Java has boolean type but C does not. Java has more data types because it is object-oriented. Therefore, java is more understandable aspect of readability. The other sub-title is **Syntax Design.** This feature is the most significant according to many people because developers use many programming languages, and they want to see similar syntax for every language. For example, Conditional statement must be **if-else** structure, Loop statement must be **While** and **For**, Function must be **function_name (variable){implementation},** calling method for object-oriented must be **class.method** etc. When we considering to our language which are advisor for other languages. Therefore, their syntax design is very sufficient. The other criteria are **Writability.** The sub-topics which are **simplicity and orthogonality** are the same with Readability. Java is more accurate for writability because it is high-level language and is an object-oriented language. The other criterion is **Reliability** depends on also writability and readability. The program must be reliable that means it performs to its specifications under all condition. It has two sub-topic which are **Type checking** and **Exception handling**. Type checking is a fundamental aspect of reliable criterion, encompassing the testing of type errors in a program, either during compilation or at runtime. Run-time type checking tends to be more resource-intensive, while compile-time type checking is generally preferable due to its efficiency. In the context of Java, the language's design necessitates comprehensive type checks for nearly all variables and expressions during compilation. This rigorous approach virtually eliminates type errors at runtime. In contrast, C adopts a more permissive type of system, leading to an increased reliance on

run-time type checking, which places a greater burden on programmers to ensure type correctness. While early error detection remains essential in C, strict compile-time checks are less common, making it possible for type errors to surface during program execution. **Exception handling's ability** of a program to intercept run- time errors (as well as other unusual conditions detectable by the program), take corrective measures, and then continue is an obvious aid to reliability. C does not have these facilities, but Java does. Therefore, we can say that Java is more reliable than C programming language. The last topic of Language evaluation criteria is **Cost**. Java tends to have a lower development cost due to strong type checking, rich libraries, and automatic memory management, making it more productive. In contrast, C development can be more time-consuming and error-prone with manual memory management and lower-level operations, leading to higher development and maintenance costs. Java's strict type system and garbage collection reduce maintenance costs, while C's manual memory management and error-prone nature can make maintenance more expensive. Performance costs vary; Java may have overhead due to the JVM and garbage collection, while C is known for high performance, albeit with added development complexity. In terms of portability, Java's platform independence reduces porting costs, while C's portability depends on coding practices and platform-specific features. The choice between them depends on the specific project needs and trade-offs between development ease and low-level control.

In this segment, we will delve into the distinctions between the Java and C programming languages regarding their approaches to **Implementation Method**. There are three implementation method which

Figure 1.3



are **Compilation**, **Pure Interpretation** and **Hybrid Implementation Systems**. programs can be translated into machine language, which can be executed directly on the computer. This method is called a **compiler implementation** and has the advantage of very fast program execution, once the translation process is complete. C use this implementation. The process of compilation and program execution are shown in Figure 1.1. The other implementation method is **Pure Interpreter** which has the advantage of allowing easy implementation many source-level debugging operations, because all run-time error messages can refer to source- level units. On the other hand, this method has the serious disadvantage that execution is 10 to 100 times slower than in compiled systems. **Hybrid Implementation Systems which i**s faster than pure interpretation because the source language statements are decoded only once. Initial implementations of Java were all hybrids. Its intermediate form, called byte code, provides portability to any machine that has a byte code interpreter and an associated run-time system. Together, these are called the Java Virtual Machine. There are now systems that translate Java byte code into machine code for faster execution. The process of compilation and program execution are shown in Figure 1.1. Briefly, C uses a compilation-based approach for fast execution but with limited debugging support, while Java employs a hybrid approach that offers portability and can achieve faster execution through just-in-time compilation. This highlights the trade-offs between execution speed and debugging capabilities in the two languages.

Figure 1.2

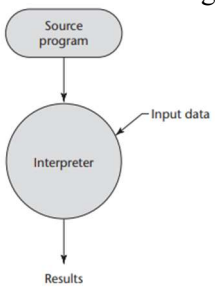
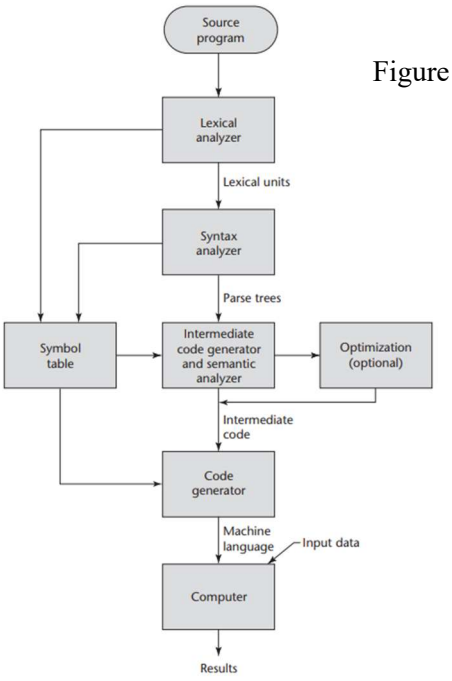


Figure 1.1



In this section, we discover on differences between C and Java aspect of the **Describing Syntax and Semantics.** Before evaluating the differences, Lets give description of Syntax and Semantics. The **syntax** of a programming language is the form of its expressions, statements, and program units. Its **semantics** is the meaning of those expressions, statements, and program units. To understand the differences, we can observe this example:

```
int x = 5, y = 0;

int result = x / y;
```

Already, evaluated implementation method and Language evaluation criteria. What we except as a behavior of Java and C for this code? Java return exception like exception:divided by zero. However C does not support object-oriented principle therefore don't return exception in C. This example shows that syntax is correct. We can divide two integers according to syntax checking but semantically it is wrong because any integer cannot be divided by zero. Therefore, Java is more reliable and more logically semantic than C. How programming language understands strings? This is a general problem of describing syntax. Firstly, to handle this problem, we use small units called **lexemes** which's description can be given by a lexical specification, which is usually separate from the syntactic description of the language. The lexemes of a programming language include its numeric literals, operators, and special words, among others. One can think of programs as strings of lexemes rather than of characters. After, description, we obviously observe that programming language must assign Lexemes to some label which is called **Tokens.** In Java statement, for example,

Lexemes	**Tokens**
index	Identifier
=	equal_sign
2	int_literal

index = 2 * count + 17;

*	mult_op
count	Identifier
+	plus_op
17	int_literal
;	Semicolon

As comparison of the lexemes and syntax of Java and C, are not quite different as a matter of fact they have very close syntax and lexeme because they are both imperative programming language which consists of a sequence of commands to alter program state even Java are object-oriented programming language. Differences are caused by difference of object-oriented, **memory management**, **Pointer manipulation**, **Code Reusability**, **Exception Handling**, **Platform dependency**. C have manual memory management with function like **malloc**, **calloc** and **free** but Java have automatic garbage collection for memory management, reducing the risk of memory leaks. C is lack of object-oriented programming, but Java fully supports OOP principles including **classes**, **inheritance**, and **polymorphism**. Also, according to **Access Specifiers** is feature of Java with keyword such as **private**, **protected**, and **public**. Java have feature of Code reusability because of OOP principle. Exception Handling is shows up thanks to OOP principle. Java prevents run-time error with built-in exception handling using **try**, **catch**, and **throw** keyword. The other significant difference of Pointer Manipulation. C allows direct manipulation of memory addresses using **pointers,** which ensure that C is a **middle-level** programming language. Also, some keywords are different such as **const** for C, **final** for Java. Even if they have close meaning but they are different and intended purpose of these are different. We can explain like that the const keyword in C is used to declare a constant value, while the final keyword in Java is used to declare a constant variable (often used in the context of classes and methods). The key difference is that const creates a constant value, and final creates a constant reference to a value. In other words, const makes the data itself

constant, whereas final makes the reference to the data constant. This kind significant differences influences the syntax and lexeme and tokens. It seems that they have lots of difference but also, they have very common syntax for example control statements keyword **if**, **elseif**, **else, swich.** Loop statements **while** and **For** loop statements and loop's keyword like **break**, **continue**. Also, they have same value type **int**, **float**, **char**, **string.** They have also same array support they reach all elements of array with **index operator [].** Also, they separate the specific field of loop or function with **curly braces {}.** They have similar syntax like these which also provides similar tokenization for identifiers, constants, special symbols like arithmetic symbols (+ - / *). **Semicolon (;)** is used to terminate the statements for two languages. These syntax's differences and similarities create semantic meaning for both two languages. These semantics meaning are quite similar but sometimes they are different because of difference semantics also different **Platform Dependency** of Java and C programming languages. Because C requires recompilation for different platforms. Java is an independent platform thanks to Java Virtual Machine (JVM) that allows run everywhere.

In the significant section, we will compare the paradigms of Java and C, delving deeper into their distinct programming philosophies. Java and C represent two distinct programming paradigms. Java is firmly rooted in the **object-oriented programming (OOP) paradigm**, with a strong focus on classes, inheritance, and encapsulation. In Java, everything is an object, and it promotes the principles of code reusability and modularity through the use of classes and objects. This OOP approach encourages developers to think in terms of objects and their interactions, making it well-suited for building complex, scalable software systems. Java's automatic memory management and robust exception handling contribute to its reputation as a reliable and secure language. On the other hand, C, while not exclusive to a single paradigm, combines elements of both **procedural and structured programming**. It provides greater control over low-level system resources and memory, making it a go-to choose for system programming and embedded systems. In C, the focus is on functions and structured programming, which

emphasizes breaking down a program into modular, reusable functions. While C can incorporate object-oriented concepts when needed, it doesn't provide the same level of support for OOP as Java. Additionally, C's manual memory management and more permissive type system give developers greater control but also require a deeper understanding of the system's intricacies. Both languages have their unique strengths, and the choice between them often depends on the specific paradigm and development needs of the project. Java excels in building robust, maintainable, and platform-independent applications, while C is favored for tasks that require fine-grained control over system resources and performance.

In the last section, we consider usable topics of programming languages such as **availability**, **efficiency**, **learning curve**. When it comes to efficiency, C often holds an advantage over Java, particularly in terms of raw computational performance. C's low-level nature allows for fine-grained control over system resources, making it a preferred choice for developing operating systems, embedded systems, and real-time applications, where high performance and minimal resource overhead are essential. In contrast, Java introduces some performance overhead due to its reliance on the Java Virtual Machine (JVM) and garbage collection. While modern Java implementations have made significant performance improvements, C remains the language of choice for applications that demand peak efficiency. In terms of availability, Java has a distinct advantage. Its "Write Once, Run Anywhere" capability, driven by the platform-independent nature of the JVM, allows Java applications to run on various platforms without modification. This significantly reduces the effort required to make software available to a wide audience. On the other hand, C may necessitate recompilation and adjustments when targeting different platforms, potentially limiting its accessibility and portability for cross-platform development. Furthermore, considering the learning curve, Java is generally considered more manageable for beginners and those transitioning from other programming languages. Its emphasis on object-oriented programming (OOP) principles and its clear, verbose syntax makes it easier for developers to grasp. Additionally, Java's intense standard library provides pre-built functions and classes for common tasks, further simplifying the development process. Conversely, C can pose a steeper learning curve for

beginners due to its low-level nature and complex memory management. Understanding concepts like pointers, memory allocation, and system resource management can be intimidating for newcomers. However, for experienced programmers, especially those with a background in C, the learning curve may not be as steep.

In conclusion, the comparison between the Java and C programming languages reveals a multitude of factors that play a crucial role in determining the suitability of each language for various development scenarios. These languages have evolved over time, each with distinct features and paradigms. Java stands out for its robust object-oriented programming approach, platform independence, and reliability through features like exception handling, making it an excellent choice for building portable and secure applications. On the other hand, C offers unmatched efficiency and low-level control over system resources, making it the preferred language for applications demanding peak performance, such as operating systems and embedded systems. The differences in readability, orthogonality, data types, and syntax design highlight the trade-offs between simplicity and flexibility. Java's greater simplicity and orthogonality, coupled with a rich standard library, ease the learning curve, and enhance writability. Moreover, the presence of comprehensive data types in Java enhances readability. However, C's unique features like direct memory manipulation through pointers and its adaptability to various platforms contribute to its strength in efficiency. Both languages differ significantly in implementation methods, where C relies on compilation for high-speed execution, and Java employs a hybrid approach with portability in mind. The differences in syntax and semantics stem from their distinct paradigms, with Java being firmly rooted in object-oriented programming and C encompassing elements of procedural and structured programming. Finally, the choice between Java and C hinges on factors like efficiency, availability, and the learning curve. Java excels in portability and availability, while C outperforms Java in efficiency. In terms of learning curve, Java is more approachable for newcomers, while C can be daunting due to its low-level nature. Ultimately, the decision depends on the project's specific needs and the developer's familiarity with each language, as well as the critical balance between the ease of

development and low-level control. These languages have left indelible marks on the world of software development, and their unique attributes make them indispensable tools for different applications.

References:

1. Robert W. Sebesta Concepts of Programming Languages
2. The C Programming Language (1st ed). Englewood Cliffs,
3. The Java Language Specification, Java SE 7 Edition

Ziya Kadir TOKLUOGLU

210104004228