Course: MSc DS

Java Programming

Module: 1

Preface

The course titled "Java Programming" aims to provide comprehensive guidance to individuals with varying backgrounds, including experienced programmers well-versed in other programming languages and novices who are taking their first steps in the realm of coding. The information is organised in a manner that facilitates a progressive and comprehensive comprehension of Java, including its underlying principles and extensive range of functionalities.

Module 1 will explore the early origins of Java, examining the underlying reasons for its development and its fundamental concepts. In this academic discourse, we will thoroughly examine the grammar and fundamental components that contribute to the potency and user-friendliness of the Java programming language, particularly for novice learners.

Module 2 explores the fundamental principles of Object-Oriented Programming (OOP) in the context of Java. In this course, students will develop the ability to conceptualise problems in terms of objects and use the fundamental concepts of Object-Oriented Programming (OOP), including inheritance, polymorphism, and encapsulation, to shape their solutions.

In the transition to Module 3, we will examine the mechanisms Java offers for managing the sequence of operations inside a programme and effectively structuring data, ranging from fundamental arrays to advanced collections. Furthermore, we

willaddress the strong error-handling methods in Java.

Module 4 of the course delves into the intricacies of Java, exploring its more complexfunctionalities. The course will cover the mechanisms through which Java communicates with external resources, controls concurrent execution, and offers tools that facilitate the development of more flexible and dynamic code.

Finally, Module 5 provides learners with an introduction to the wider Java environment. In this program, participants will have the opportunity to get practical experience with widely-used frameworks and tools, therefore enhancing their proficiency as Java developers.

Learning Objectives:

1. Understand Java's beginnings and development while acknowledging itsimportance in the programming community.
2. Gain a solid understanding of Java's basic grammar and structure to set the stagefor more complex subjects.
3. Establish a foundation for data manipulation by becoming familiar with the mainvariables and data types in Java.
4. Understand the fundamental operatorsand expressions, and practise input/output procedures on a real console.

Structure:

1.1 History of Java

The computer language Java, which is renowned for its versatility and significant impact, can be traced back to the early 1990s. The conception of this project was initiated by a team under the leadership of Dr James Gosling at Sun Microsystems, a renowned corporation known for its expertise in hardware systems. The primary objective was to create a programming language specifically designed for digital devices, including set-top boxes and TVs. The project, known as "Green," did not instantly accomplish its stated objective, but it established the groundwork for the emergence of Java.

The programming language was first designated as "Oak," drawing inspiration from an oak tree situated next to Gosling's workspace. However, in 1995, it underwent a name change to "Java," obtaining its nomenclature from the renowned Java coffee. The primary driving force behind the creation of Java was the guiding philosophy known as "Write Once, Run Anywhere." This implied that programmers had the ability to create Java code once and execute it on any device equipped with a Java Virtual Machine (JVM). The concept of universality presented a resolution to the prevalent fragmentation challenges seen in the business when software needs rewriting to accommodate various platforms.

The first public release of Java occurred in 1996, marked by the introduction of its inaugural version, Java 1.0. The technology

quickly garnered attention, particularly in the realm of online platforms, as a result of its commitment to providing safe, platform-agnostic, and network-savvy apps. The incorporation of applets facilitated the integration of interactive content into web browsers, resulting in a significant increase in the popularity of Java.

Throughout its existence, the Java programming language has had several iterations, including various enhancements that have ensured its relevance and significance within the dynamic technological environment. Following the purchase of Sun Microsystems by Oracle Corporation in 2010, Java was successfully integrated into Oracle's portfolio, allowing it to continue its tradition of pioneering advancements in the field.

1.2 Basic Syntax and Structure

Java, like other high-level programming languages, depends on a distinct grammar and structure that serve as the foundation for developers to compose efficient programs. Gaining a thorough understanding of these foundational principles is crucial for anybody embarking on the study of Java programming.

The fundamental element of Java's syntax is the class. Each Java programme begins with a minimum of one class, which serves as a container for both data and behaviours. The class is declared with the class keyword, followed by its identifier. An example may be seen in the naming convention of a class called

"HelloWorld," which is denoted by the declaration "class HelloWorld."

The main method is often seen as the principal function inside this class. The method in question functions as the designated starting point for Java programs and isdenoted by the following syntax: public static void main(String[] args). The terms "public" and "static" indicate that the method is accessible from any location and does not need an instance of an object, respectively. The void keyword is used to signify that a procedure does not provide any output.

The structure of Java is characterised by a high degree of modularity, whereby code is systematically organised into distinct units known as methods. These methods are designed to carry out certain tasks and may be called or triggered as needed. In the Java programming language, it is customary for each statement to conclude with a semicolon (;) as a means of indicating the termination of a specific command.

In the Java programming language, the process of declaring a variable necessitates the inclusion of a type definition, hence enforcing the principle of strong typing. To illustrate, in order to define an integer variable denoted as 'age', it is necessary to write: int age; The grammar of Java exhibits rigorous adherence to data types, hence minimising the occurrence of runtime errors and enhancing code clarity.

The use of indentation and white spaces, while not mandated

by Java's grammar, plays a crucial role in enhancing the readability of code. The practice of maintaining consistent indentation in programming aids developers in comprehending the logical progression of programs and facilitates differentiation between various code blocks, particularly those included inside conditional statements and loops.

1.3 Data Types and Variables

In the context of programming, Java is classified as a statically-typed language, which implies that it requires explicit declaration of the data type for each variable prior to its use. The inclusion of data type specifications from the start of a program guarantees type safety, resulting in a reduction of runtime mistakes associated with types and a more comprehensible comprehension of variable use.

The fundamental building blocks of Java are its basic data types. The data types included in this category consist of byte, short, int, and long for representing integer values; float and double for denoting floating-point numbers; char for representing characters; and boolean for indicating true/false values. Each of these classifications has a predetermined magnitude and scope. For example, in the Java programming language, an integer (int) generally occupies a memory space of 4 bytes and has the capacity to represent numerical values ranging from -2^31 to 2^31-1.

Subsequently, the reference data types include objects and

arrays. In contrast to basic types, reference types are derived from classes and interfaces, indicating memory locations where data is kept rather than directly encapsulating values.

In the Java programming language, a variable is a designated region of memory that is assigned a name and is capable of storing data. In order to define a variable, one must provide its data type, followed by its designated name. An example of declaring an integer variable called "age" may be seen in the statement "int age;". In programming, variables have the option to be initialised upon declaration, such as "int age = 25," or to set a value at a later point in the programme.

The selection of a data type is of utmost importance as it dictates the dimensions and arrangement of the storage for a variable, the scope of values that may be kept inside it, and the permissible actions that can be executed on it. Moreover, the robust type-checking System in Java during compilation guarantees that a variable does not exhibit unexpected behaviour, such as placing a string value in a variable of integer type.

1.4 Operators and Expressions

Java provides a comprehensive range of operators that enable programmers to execute diverse operations on variables and values. Operators are symbolic representations that direct the compiler to do certain mathematical, relational, or logical calculations. Expressions in Java are formed when operators

are used in conjunction with variables and literals. These expressions serve as fundamental components for many programming structures.

Arithmetic operators are widely used in mathematical procedures, making them the most prevalent. The arithmetic operations included in this set are addition (+), subtraction (-), multiplication (*), division (/), and modulus (%), which are used to determine the residual resulting from a division operation. As an example, the statement "int result = 10 * 3" employs the multiplication operator to perform the multiplication operation between the operands 10 and 3.

Relational operators facilitate the comparison of two values and ascertain their relative connection. The operators encompassed in this set are equality (==), inequality (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). For instance, the conditional statement (a > b) would evaluate the inequality between the values of a and b, determining if an is bigger than b.

Logical operators are used in programming to facilitate decision-making processes. The fundamental logical operators include AND (&&), OR (||), and NOT (!). Boolean expressions are manipulated by these operators, which produce boolean outcomes. Such an instance is a conditional expression (isTall && isSmart), which evaluates if both criteria are simultaneously true.

Furthermore, inside programming languages, there are assignment operators, such as the "=" operator, which serves the purpose of assigning a specific value to a variable. Additionally, compound assignment operators, exemplified by "+=" or "*=", are used to combine an arithmetic operation with the assignment of a value to a variable.

Unary operators perform operations on a singular operand. The increment (++) and decrement (--) operators, for example, are used to augment or diminish a value by one, correspondingly.

1.5 Simple Console Input/Output

The Java programming language offers a simple and direct approach to facilitating user interaction via console-based input and output functions. This interaction is crucial for fundamental applications, the process of identifying and resolving errors, or for the development of interfaces that rely on textual input and output.

In the context of console output, the major mechanism used by Java is the System.Out object, which is an inherent utility inside the Java programming language. Thisobject serves the purpose of facilitating the publishing of textual information to theconsole. The println() function is often used for this object since it facilitates theprinting of a line of text into the console, followed by a subsequent line transition. Anexample of a code snippet that demonstrates the display of the greeting "Hello,World!" on the console is System.

Out.println("Hello, World!");. Furthermore, theprint() function may be used to output text without initiating a line break thereafter. In contrast, the process of console input is somewhat more complex but still feasible.Java provides the Scanner class, which is included in Java. Util the package to easethis process. Prior to using the Scanner, it is essential to import it by including thestatement "import java.util.Scanner;". After being imported, it is possible to instantiate a Scanner object in order to get input from several sources, such as theconsole.

1.6 Writing and Running a Simple Java Program

The initiation of the Java programming endeavour commences with comprehending the procedure of composing and running a rudimentary Java program. This first stage serves as a fundamental step, shedding light on the trajectory towards more intricate applications and features inside the Java ecosystem.

In order to begin the process of building a Java program, it is necessary to possess a rudimentary text editor or an integrated development environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans. Integrated Development Environments (IDEs) often provide inherent functionalities that enhance the efficiency of creating, debugging, and executing Java programs. Nevertheless, novice individuals may find it advantageous to

comprehend the manual procedure.

Typically, a basic Java programme starts with a declaration of a public class. It is necessary for the class name to correspond with the filename. As an example, if the file is denoted as HelloWorld.java, the class therein should be designated as HelloWorld. In the context of this course, the public static void main(String[] args) method serves as the designated entry point for the Java programme. The programme execution starts at this point, namely inside the Java Virtual Machine (JVM).

1.7 Summary

- Module 1 offers a thorough overview of the Java programming language, providing an extensive introduction to its fundamental concepts and principles. Commencing with an overview of the historical background of Java, this elucidates the development and importance of a widely acclaimed programming language that has attained global popularity.
- The module further explores the fundamental principles of the Java language, with particular emphasis on its fundamental grammar and structural components. This is an in-depth examination of the several data types in Java and the significance of variables in the storage and manipulation of data. Moreover, the module provides a comprehensive explanation of the fundamental principles behind operators

and expressions, emphasising their role in enabling computational operations and data manipulations inside Java programming. An essential element of this session is showcasing basic console-based input and output processes, which aim to enable learners to effectively engage with users and provide outcomes.

- Ultimately, the module concludes by providing participants with guidance on the process of composing and implementing a rudimentary Java programme, establishing the foundation for more sophisticated programming techniques in later modules. Module 1 serves as a foundation for acquiring the essential information necessary to effectively traverse the vast and ever-evolving domain of Java programming.

1.8 Keywords

- **Java:** One notable programming language that exhibits versatility and follows an object-oriented paradigm is renowned for its ability to be easily deployed across several platforms.
- **Syntax:** The collection of regulations and customary practises that provide the framework and organisation of a programming language.
- **Data Types:** Data categories are used to establish the specific kind and characteristics of items that may be saved. These categories include integers, floating-point numbers,

and characters.

- **Variables:** Storage sites in a program refer to specific memory regions that are used to store data values. These storage locations have the capability to be modified throughout the execution of the programme.
- **Operators:** Symbols are used in mathematics, as well as in relational and logical contexts, to carry out operations on variables or values.
- **Console I/O:** The process of obtaining user input and presenting output on the console is often achieved via the use of classes such as Scanner for input acquisition and System, out for output presentation.

1.9 Self-Assessment Questions

1. Outline Java's historical turning points and the effects they have on the software development sector.
2. Describe the fundamental grammar and organisation of a Java programme and why it is crucial for maintainability and code readability.
3. Compare and contrast Java's primitive and reference data types. Give illustrations of each.
4. Describe how Java operators are used. What are the functional differences between arithmetic, relational, and logical operators? Give an example of each.
5. Describe how to utilise the console in Java to get user

input and show output. What types of courses and techniques are often used to achieve this goal?

1.10 Case Study

Title: The Evolution and Implementation of the Java Programming Language at XYZ Corporation

Introduction:

XYZ Corporation, a firm operating in the field of information technology, has historically relied on obsolete programming languages for its internal software applications. In order to remain abreast of contemporary technological advancements and enhance operational efficacy, the organisation was contemplating a shift to the Java programming language. The individual or group expressed a need for a language that has qualities of robustness, modernity, and scalability in order to provide a solid groundwork for future endeavours.

Case Study:

Over an extended period, XYZ Corporation used outdated systems, sometimes referred to as legacy systems, that were progressively becoming burdensome and incompatible with emerging technology. Following an internal evaluation, the management acknowledged the need for a comprehensive revision of the terminology used. Java has gained prominence as a leading candidate because of its recognised attributes of adaptability, resilience, and scalability.

Background:

Java, renowned for its "write once, run anywhere" paradigm and adherence to object-oriented concepts, has emerged as a prevailing entity within the realm of software development. The history of this subject is characterised by a series of ongoing improvements, which have ensured its continued relevance even many decades after its first appearance. Due to its extensive functionalities and widespread use, several enterprises have transitioned to Java as their preferred platform for software development requirements.

Your Task:

You are appointed as the Lead Software Developer at XYZ Corp. The main objective is to develop a rudimentary Java-based application that will serve as a replacement for one of the existing legacy systems. The programme needs to exhibit the fundamental concepts of Java, including its grammar and underlying features.

Questions to Consider:

1. How would one strategise the transition process from the current System to the newly implemented Java-based application?
2. In the application, it is crucial to highlight the essential characteristics of Java that showcase its superiority over the traditional System.
3. Given the historical context and evolutionary trajectory of

Java, what measures would you implement to guarantee the application's scalability and adaptability in light of future Java updates?

4. How can Java's console input/output capabilities be effectively used to enhance the user-friendliness of the programme for workers of XYZ Corp?

Recommendations:

It is advisable to start with a trial initiative, prioritising one of the legacy systems that have lesser significance. Through the use of Java's comprehensive grammar, efficient data manipulation, and input/output functionalities, the pilot programme may function as a prototype for subsequent, more intricate transitions. It is recommended that regular training sessions be conducted to acquaint the personnel with the operational aspects of Java.

Conclusion:

The extensive historical background and multifaceted capabilities of Java provide it a prudent selection for XYZ Corp's transitional endeavours. By adopting a strategic methodology and prioritising the inherent advantages of Java, the organisation may establish a foundation for a future characterised by enhanced technical capabilities and improved operational efficiency.

1.11 References

1. Alpern, B., Attanasio, C.R., Cocchi, A., Lieber, D., Smith, S., Ngo, T., Barton, J.J., Hummel, S.F., Sheperd, J.C. and Mergen, M., 1999. Implementing jalapeño in java. ACM SIGPLAN Notices, 34(10), pp.314-324.
2. Deitel, P.J., 2002. Java how to program. Pearson Education India.
3. Schildt, H., 2003. Java™ 2: A Beginner's Guide.
4. Edelstein, O., Farchi, E., Nir, Y., Ratsaby, G. and Ur, S., 2002. Multithreaded Java program test generation. IBM systems journal, 41(1), pp.111-121.
5. Eckel, B., 2003. Thinking in JAVA. Prentice Hall Professional.