Course: MSc DS

Java Programming

Module: 3

Learning Objectives:

1. Understand and use different conditional statements to control how a Java programme executes.
2. Understand how to use various loop structures in Java for iterative processes.
3. To manage and organise data, be familiar with the foundations of arrays and ArrayLists.
4. Understand more complex collections like Maps and Sets, and become an expert at handling mistakes and exceptions in Java programs.

Structure:

3.1 Conditional Statements

In the domain of programming, the act of making choices has significant importance. Similar to people, computers assess situations in order to select the future course of action. Java, like several programming languages, uses conditional expressions to assist the process of making judgements. The fundamental conditional statements in the Java programming language consist of the "if", "else", and "switch" components. The "if" statement is the most fundamental kind of conditional evaluation. The programme scrutinises a condition, and if the condition is determined to be true, a certain set of instructions is performed. Conversely, if the condition is deemed false, the programme will skip over the corresponding block of code. In the context of a meteorological application, an "if" statement may be used to evaluate the presence of rainfall. If the statement is accurate, the programme may recommend the action of bringing an umbrella.

Expanding upon the conditional statement, the "else" clause is introduced. The "if" statement evaluates a condition, whereas the "else" statement encompasses all other scenarios that do not satisfy the original condition. Let us revisit the weather application scenario: in the event that there is no occurrence of rainfall, denoted by the false evaluation of our conditional statement, the subsequent "else" phrase may propose the use of sunglasses as a suitable course of action.

In contrast to the binary conditions handled by "if" and "else" statements, real-world circumstances often include the presence of many conditions. The "switch" statement is used in this context. The feature enables the evaluation of a variable against several values. In this context, every individual value is referred to as a "case", and the variable is systematically tested against each case until a corresponding match is identified. As an example, in an application that involves the determination of days of the week, Monday may be designated as case 1, Tuesday as case 2, and so on. The "switch" line subsequently evaluates the value of the day and performs the corresponding case.

3.2 Loops

In the field of programming, it is common to encounter situations that need the execution of repeated actions. These actions may include tasks such as computing the sum of integers inside a given list or processing many lines of data from a file. In order to avoid repetitive code, loops are used. The Java programming language has three main types of loops: "for", "while", and "do-while". These loops provide distinct benefits that cater to certain contexts.

The "for" loop is a well-defined and often used construct in programming, particularly when the total number of iterations is predetermined. The structure has three components, namely initialisation, condition, and iteration. For example, if an

individual wants to output a sequence of values ranging from 1 to 10, a "for" loop is a suitable construct to use. The loop starts by initialising a counter, thereafter evaluating it against a condition to see whether it is less than or equal to 10. Following each iteration, the number is incremented. The methodical nature of this technique renders "for" loops well-suited for the purpose of iterating across arrays or collections with a predefined size.

In contrast, the "while" loop exhibits a greater degree of flexibility and indefinite iteration. Prior to executing the contained statements, it verifies a condition. In the event that the condition evaluates to true, the loop will execute; otherwise, it will terminate. This iteration structure is particularly suitable in situations when the precise number of iterations cannot be predetermined. Let us contemplate a hypothetical situation in which a computer programme continuously receives user input until the user chooses to terminate the program. A "while" loop may be used to repeatedly evaluate if the user's input does not correspond to the "quit" command, and if this condition is met, the programme will proceed with further processing.

Finally, the "do-while" loop presents a minor variation of the "while" loop. The loop's body is guaranteed to execute at least once prior to evaluating the condition. This practice is advantageous in cases when a certain operation must be

performed before the initial condition is evaluated. In a menu-driven application, it is common practice to provide the menu to the user first and afterwards verify their selection against the available alternatives.

3.3 Arrays and ArrayLists

In the Java programming language, the task of arranging and overseeing various data elements often requires the use of arrays and ArrayLists. Both structures are essential, albeit they serve different purposes and have their own distinct advantages and disadvantages.

Arrays are considered to be essential data structures in the Java programming language since they serve as containers that hold a defined number of items, all of which must be of the same data type. Upon the establishment of an array, its size is established and stays immutable, hence prohibiting the addition or removal of items subsequent to its initialisation. The inherent immutability of arrays results in their memory efficiency since their dimensions are predetermined. Nevertheless, the fixed size of the data structure might provide a constraint when there is ambiguity over the number of pieces that need to be stored. An example of an integer array that stores five scores may be shown with the declaration: int[] scores = new int[5];. Accessing or modifying individual scores can be achieved by using an index, such as scores[2] = 85; In contrast, ArrayLists are a component of Java's Collections

Framework and provide the capability for dynamic scaling. In contrast to arrays, ArrayLists possess the capability to dynamically adjust their size, expanding or contracting as required. This attribute renders them well-suited for situations where the number of components cannot be accurately anticipated. The ArrayList is implemented using an underlying array structure, which allows for automated enlargement when the number of elements exceeds the existing capacity or when elements are deleted. In order to use an ArrayList, it is necessary to import the Java. Util package. The inherent variability of dynamic arrays, while providing adaptability, might result in significantly slower performance compared to normal arrays owing to the additional computational burden of handling the varying size. An example of using an ArrayList to hold names would be as follows: ArrayList<String> names = new ArrayList<>();. To add names to this ArrayList, the names may be appended using the names. The user has requested to add the string "Alice" to the existing data.

Arrays and ArrayLists are fundamental components for data management in the Java programming language. Arrays are known for their compactness and efficiency when dealing with collections of constant size. However, ArrayLists have the advantage of dynamic resizing, making them very helpful in scenarios where the size of the dataset may vary throughout the execution of a programme.

3.4 Maps, Sets, and Other Collections

The Collections Framework in Java is an extensive collection of data structures that offers a wide range of capabilities for the storage and organisation of data. The toolkit comprises crucial components such as Maps, Sets, and various collection types like Queues and Deques.

Maps in the Java programming language are essential data structures that facilitate the association of distinct keys with corresponding values. The concept has a resemblance to that of a tangible lexicon, whereby one consults a term (the key) in order to ascertain its meaning (the value). The HashMap is a widely used implementation of the Map interface, providing average constant-time performance for fundamental operations. As an example, in the case of desiring to generate a telephone directory, one might use a HashMap<String, String> data structure, whereby the names would serve as the keys, and the corresponding phone numbers would be designated as the related values.

Sets, on the other hand, are specifically structured to include a distinct assemblage of items, hence precluding any instances of duplication. The HashSet is a widely used implementation that does not preserve any particular order of its components. If one wants to monitor a collection of distinct student identification numbers, a HashSet<Integer> would be appropriate. In

situations when the preservation of insertion order is crucial, the Java programming language provides the LinkedHashSet data structure.

In addition to Maps and Sets, the Collections Framework in Java offers specialised data structures such as Queues and Deques. A Queue is designed to adhere to the FIFO (First-In-First-Out) concept, which makes it well-suited for applications such as job scheduling. The Deque, also known as a double-ended queue, is a highly adaptable data structure that allows the addition or removal of components from either end. It serves a wide range of functions, including simple lists as well as intricate algorithms.

Java provides a wide range of collection types that are designed to meet particular requirements for data management. Maps provide the establishment of relationships between keys and values, Sets guarantee the absence of duplicates, and other collections such as Queues and Deques offer more versatility. Collectively, these components serve as the fundamental infrastructure for the storing and manipulation of data inside programmes developed using the Java programming language.

3.5 Exceptions and Error Handling

In the realm of programming, it is not uncommon for deviations from intended outcomes to occur. During the execution of a programme, errors may occur, leading to disruptions in its flow.

Java, like other contemporary programming languages, provides a comprehensive method to handle these interruptions effectively, namely via the use of exceptions and error management.

The fundamental principle underlying Java's error-handling mechanism is the notion of an "exception". An exception may be defined as an occurrence that emerges inside the course of programme execution, causing a disruption to its typical progression. Consider it as an indication that an anomaly has occurred, including scenarios such as an erroneous input, an inaccessible file, or a disrupted network connection.

Java classifies exceptions into two primary categories: verified exceptions and unchecked exceptions. Checked exceptions are indicative of circumstances that a meticulously crafted programme ought to foresee and rectify. An instance of a file not being found is sometimes seen as a checked exception, specifically the FileNotFoundException. In contrast, unchecked exceptions often indicate programming errors, such as attempting to access an erroneous index of an array (ArrayIndexOutOfBoundsException).

The programming language provides the try-catch construct as a means of handling these exceptions. Code that has the potential to generate an exception is often included inside a try block. In the event that an exception occurs, the programme promptly transitions to the associated catch block, where the

exception is intercepted and managed. This feature guarantees the prevention of unexpected programme crashes, enabling it to either recover smoothly or provide the user with appropriate notifications on the encountered problem.

In addition, Java has a final block that is executed unconditionally after the try-catch block, regardless of whether an exception was raised. The purpose of this block is often to do cleaning tasks, such as finalising opened files or terminating network connections.

Nevertheless, it is important to use prudence while using exceptions. Excessive dependence on them might result in decreased code readability and diminished efficiency. The crucial aspect is in using them in scenarios where exceptional circumstances truly arise rather than for routine control flow inside a programme.

3.7 Summary

- Java provides a rich range of control structures and collection types that are essential for efficient programming. Java has conditional statements such as "if", "else", and "switch" to streamline decision-making processes inside code. These structures enable developers to control the flow of a programme depending on criteria, ensuring that certain pieces of code are executed only when intended.
- Furthermore, loops like "for", "while", and "do-while"

facilitate the iterative execution of code blocks, hence enabling smooth activities such as iterating through lists or repeatedly evaluating conditions. Every form of loop has distinct advantages that cater to various programming requirements.

- In the realm of data storage, Java offers straightforward data structures such as Arrays, as well as flexible ones like ArrayLists. Arrays are characterised by a fixed size and are known for their speedy and efficient performance. On the other hand, ArrayLists possess the advantageous feature of dynamic resizing, enabling them to adapt to variable dataset sizes and therefore offering more flexibility.
- In addition, Java has more sophisticated data structures, such as Maps and Sets. Maps are data structures that store key-value pairs, facilitating efficient data retrieval. On the other hand, Sets are data structures that store unique values, which is useful for operations such as deleting duplicates. These collections, in addition to others such as Queues and Stacks, provide diverse possibilities for manipulating data.
- Finally, the module explores the concepts of exceptions and error management. The Java programming language employs "try-catch" structures as a means of handling unforeseen events that may occur during programme execution. This approach serves to reduce the risk of programme crashes and promotes more seamless

functioning of software systems.

- This module provides a fundamental understanding of Java's control and data structures, facilitating proficient and impactful programming practices.

3.8 Keywords

- **Conditional Statements:** The programme flow is directed depending on specified circumstances.
- **Loops:** Enables the iterative execution of code portions.
- **Arrays:** Fixed-size data storage structures are used for storing homogeneous items.
- **ArrayLists:** Dynamic data storage structures are data structures that have the ability to increase or decrease in size.
- **Maps:** Key-value pair collections are used to store data in a manner that facilitates efficient retrieval.
- **Exceptions:** Unforeseen occurrences that arise during the execution of a programme and need appropriate management or handling.

3.9 Self-Assessment Questions

1. Distinguish between "if" and "switch" conditional expressions and provide an instance of when using "switch" might be more effective.
2. Describe how "do-while" loops vary from "for" and "while" loops and provide an example of why one might be more

advantageous.

3. Evaluate and contrast Java's arrays and array lists. What benefits do ArrayLists provide over conventional arrays?
4. Describe how Java's Maps might facilitate effective data retrieval. How is a Map's structure different from a List or Set?
5. Go through the significance of Java's exception handling. Give an example of how a programme could suffer serious problems if an exception is not handled.

3.10 Case Study

Title: Efficient inventory management with Java control structures and collections. Introduction:

The effective management of inventory plays a crucial role in the contemporary retail sector, as it is essential for achieving profitability and maintaining client happiness. Efficient systems play a crucial role in the tracking of products, monitoring of sales, and prediction of future stock needs.

Case Study:

BrightMart, an expanding retail chain, has encountered difficulties with its current inventory management system. The outdated software often encounters delays, mistakes, and even data loss, therefore impacting company operations and eliciting consumer dissatisfaction.

Background:

BrightMart initially operated as a modest community-based

establishment but has since seen substantial growth, resulting in the establishment of several branches over the course of the last five years. The current inventory system in place was initially developed to accommodate the organisation's original size and lacks the efficient use of contemporary programming features such as loops, arrays, and exception handling. Consequently, the task of maintaining stocks across several locations or predicting future requirements based on sales trends is progressively getting more burdensome.

Your Task:

In the capacity of a senior Java developer, you have been engaged to undertake a comprehensive renovation of BrightMart's inventory management system. The primary task assigned to you is to devise and execute a novel system by effectively utilising Java's control structures and collections. The proposed system should effectively and smoothly facilitate the updating of inventories, the management of stock across many branches, and the resolution of possible data conflicts while maintaining stability and avoiding system crashes.

Questions to Consider:

1. How may Java's loop structures be used to automate inventory audits and modifications?
2. Which Java collections are most appropriate for storing product data, branch information, and sales records?
3. How can the conditional statements in Java be used to

effectively handle stocknotifications by considering inventory thresholds?

4. In which situations inside the inventory system would the implementation ofexception handling be of utmost importance?

Recommendations:

The integration of Java's "ArrayList" may be used to create dynamic inventory lists, which possess the ability to expand in size when new goods are introduced. The use of "Maps" facilitates the association of product IDs with their corresponding details, enabling efficient retrieval of information. Incorporate iterative structures to streamline recurring activities, such as stock checking, while using conditional statements to prompt replenishment notifications. In conclusion, it is essential to include strong exception-handling procedures in order to enhance the system's ability to withstand unforeseen data-related challenges.

Conclusion:

By using Java's control structures and collections, BrightMart has the potential to significantly transform its inventory management system. This implementation would result in enhanced efficiency, accuracy, and scalability, aligning with the company's fast expansion. The use of these technologies and processes is expected to enhance operational efficiency and enhance customer satisfaction.

3.11 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., 2003. 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., 2003. Multithreaded Java program test generation. IBM systems journal, 41(1), pp.111-123.
5. Eckel, B., 2003. Thinking in JAVA. Prentice Hall Professional.