Lec 21 String:

This lesson explains the importance and behavior of strings in Java, focusing on how strings are stored, their types (mutable and immutable), and why understanding these concepts is crucial for programming. It also highlights the difference between logic building and concept building in learning Java, and introduces key string methods for real-world applications.

Concept vs. Logic Building in Java

Learning Java requires both logic building (problem-solving ability) and concept building (understanding how things work). Logic building helps solve algorithmic problems, while concept building is about knowing how to use features like classes, objects, and arrays correctly.

Logic Building is a Long-term Process

Logic building is an ongoing process that improves with practice. Even if concept knowledge is strong, many learners struggle with logic-based problems like array manipulation and loops.

Concept Building for Java Technologies

As you progress to advanced Java topics (like Hibernate, JDBC, and Spring Boot), strong concept building becomes more valuable than logic building, especially in service-based companies where product design is prioritized over complex algorithms.

Product vs. Service-Based Companies

Service-based companies focus on practical application and product design, requiring good understanding of concepts. Product-based companies (like Google) emphasize logic and data structures/algorithms, often testing with challenging problems.

Introduction to Strings in Java

In Java, a string is a sequence of characters enclosed in double quotes. Strings are used extensively in applications for storing names, addresses, and other text data. Strings in Java are objects, not primitive types.

Types of Strings: Mutable vs. Immutable

Java has two main types of strings: mutable (can be changed, like StringBuilder and StringBuffer) and immutable (cannot be changed, like the standard String class). Immutable strings, once created, cannot be modified; any change creates a new string object.

Creating Strings in Java

Strings can be created in two ways: with the new keyword (e.g., new String("Raju")) or without it (e.g., "Raju"). Both create immutable strings, but they are stored differently in memory.

String Memory Management: Constant Pool vs. Heap

Java stores strings in two memory areas within the heap: the constant pool and the non-constant pool. Strings created without the new keyword are stored in the constant pool, allowing memory efficiency by reusing identical strings. Strings created with new always get a new memory space in the non-constant pool, even if the value is the same.

Case Sensitivity and Memory Efficiency

The constant pool only reuses strings that are exactly the same, including case sensitivity. If a string with the

same value but different case is created, it will occupy new memory.

Using the `new` Keyword for Strings

Creating strings with the new keyword always allocates new memory in the non-constant pool, even for duplicate values, which is less memory efficient but provides flexibility for runtime string creation.

Reference Comparison vs. Value Comparison

Using == compares string references (memory addresses), not actual values. Only strings from the constant pool with the same value will have the same reference. To compare string values, use the .equals() method.

Introduction to String Methods

Java's String class includes many methods for manipulating strings, such as concat() (joining strings) and equals() (comparing values). These methods are essential for tasks like counting characters, validating passwords, and processing text.

The charAt Method

charAt(index) retrieves the character at a specific position in a string (indexing starts at 0). It's useful for extracting or processing individual characters, such as when checking for palindromes or iterating through a string.

Converting String to Character Array

A string can be manually converted to a character array by iterating through each character and storing it in an array. Alternatively, Java provides the toCharArray() method to do this automatically.

Checking String Start and End: startsWith and endsWith

The startsWith() and endsWith() methods check if a string begins or ends with a specified substring. They return a boolean value and are useful for validation, such as email domain checks.

String Comparison: equalsIgnoreCase

equalsIgnoreCase() compares strings while ignoring case differences, ensuring that "RAJU" and "raju" are

considered equal. This is important for user validation where input case may vary.

Changing String Case: toLowerCase and toUpperCase

toLowerCase() and toUpperCase() convert all characters in a string to lower or upper case, respectively. This helps standardize input for comparison or storage.

Splitting Strings: split Method

The split() method divides a string into an array of substrings based on a delimiter (like spaces), enabling word counting or tokenization. The result is a string array containing each piece.

Getting Unicode Values: getBytes Method

The getBytes() method converts a string into a byte array, representing the Unicode values of each character. This is useful for encoding, data transfer, or cryptographic operations.

Palindrome Checking with Strings

A palindrome is a word or phrase that reads the same forwards and backwards (e.g., "madam", "dad"). To check for a palindrome, reverse the string and compare it to the original using string methods.

Character Shifting Challenge (Caesar Cipher)

Given a string and a shift value, each character is shifted forward by the specified number of positions in the alphabet (wrapping around if needed). This is a simple example of encryption (Caesar cipher).

The Substring Method

The substring() method extracts a part of a string based on index positions. Giving one index returns the string from that point to the end; giving two indices returns the part between them, not including the character at the second index.

The Contains Method

The contains() method checks if a specific sequence of characters exists in a string, returning true if the exact sequence is found, and false otherwise. The sequence must appear together and in the same order.

The IndexOf Method

The indexOf() method returns the position of the first occurrence of a character or substring in a string. If the item is not found, it returns -1. This helps in locating specific parts within a string.

The LastIndexOf Method

The lastIndexOf() method finds the position of the last occurrence of a character or substring. This is useful when the same substring appears multiple times and you need the last one.

The Replace Method

The replace() method substitutes all occurrences of a target substring with another substring, returning a new string with the changes made.

The Trim Method

The trim() method removes extra spaces from the beginning and end of a string. This is especially important when processing user input to ensure data is clean.

The CompareTo Method

The compareTo() method compares two strings lexicographically (based on Unicode values). It returns 0 if the strings are equal, a negative number if the first is less, and a positive number if the first is greater. This is useful for sorting and ordering strings.

The first difference got in characters in returned in this method.

The Equals Method

The equals() method checks if two strings have exactly the same characters in the same order, returning true or false. It is a straightforward way to test string equality.

Recap and Further Learning

Practicing these string methods helps in understanding when and how to use them efficiently. Some methods, like intern(), are advanced and often discussed in interviews.

Taking String Input in Java

Java uses the Scanner class to take input from users. For single-word input, the next() method is used, while for full-line input (including spaces), the nextLine() method is used. The difference is important for correctly capturing user data.

Difference Between `next()` and `nextLine()`

The next() method reads input until it encounters a space, so it only captures the first word. The nextLine() method reads the entire line, including spaces, until the user presses enter.

The Input Buffer Issue

When you use methods like nextInt() or next(), they read only the intended value and leave the newline character (\n) in the input buffer. If you call nextLine() after such methods, it may read this leftover newline as an empty input, causing unexpected behavior.

Understanding the Input Buffer Mechanism

The input buffer temporarily stores everything typed by the user. Methods like nextInt() consume only the

number, not the newline. This leftover newline can interfere with subsequent input operations, especially with nextLine().

Solution for Buffer Problems

To avoid issues when switching from numeric input to string input, insert an extra nextLine() call to consume the leftover newline character before reading the actual string input.

Where User Input Strings Are Stored

Strings entered by the user at runtime are stored in the Java heap's non-constant pool, not in the string constant pool. This avoids the performance cost of repeatedly checking the constant pool for duplicates among dynamic user inputs.

String Constant Pool vs. Non-Constant Pool

String literals (fixed values written in code) are stored in the string constant pool, while dynamically created strings (like user input or results from string methods) are stored in the non-constant pool. Constant pool lookups are slow if overloaded.

String Literals and Manipulated Strings

Compile-time string literals go into the constant pool. Strings created through operations like concatenation with the + operator may still go into the constant pool if both operands are literals, but strings created by methods (like concat(), replace(), etc.) or user input are stored in the non-constant pool.

Creating Strings with the `new` Keyword

Using new String("example") always creates a string in the non-constant pool, even if the content matches a string literal. This is rarely used in modern code but is common in legacy code and internal Java methods.

Moving Strings to the Constant Pool with `intern()`

The intern() method allows you to move a string from the non-constant pool to the constant pool. If the string already exists in the constant pool, it returns the reference to the pooled string. This can save memory if the same string is used often, but overusing it can slow down the program due to pool lookups.

Immutable strings are made in constant pool and mutable strings(builder/buffer) are made in non constant pool.

If s1 and s2 is pointing to “raju” in const pool. Suppose s1 is mutable and Suppose s1 is changed as → s1 = s1.concat(k); then changes will reflect in s2 also so that’s why immutable strings are made in const pool.

Mutable Strings: StringBuffer and StringBuilder

Java's StringBuffer and StringBuilder classes create mutable strings, meaning their content can be changed without creating new objects. Unlike String, these are not stored in the constant pool and are used for efficient string manipulation.

StringBuffer(Synchronized) vs. StringBuilder(Non Synchronized)

Both classes provide similar functionality for mutable strings, but StringBuffer is thread-safe (synchronized), making it suitable for multi-threaded environments, while StringBuilder is faster but not thread-safe.

Synchronization Explained

Synchronization ensures that only one thread can access a resource at a time, preventing data corruption in multi-threaded programs. StringBuffer is synchronized, so it's safer in concurrent scenarios, while StringBuilder is not, making it faster for single-threaded use.

Additional StringBuffer Methods

StringBuffer (and StringBuilder) offer methods like reverse(), delete(), and replace() for advanced string manipulation, making them versatile for many programming tasks.

StringBuffer and StringBuilder in Java are used to create and modify strings that can change (mutable strings). StringBuffer is synchronized (safe for use by

multiple threads at once), while StringBuilder is not, making StringBuilder faster but only suitable for single-threaded situations.

StringBuffer and StringBuilder Introduction

Mutable strings in Java can be created using StringBuffer or StringBuilder. Both allow modification of string content without creating new objects for every change. StringBuffer is thread-safe (synchronized), while StringBuilder is not.

Basic Methods of StringBuffer and StringBuilder

Common methods include append (adds text), insert (adds text at a specific position), replace (changes part of the string), delete (removes part of the string), length (returns the length), and reverse (reverses the string). These methods work similarly for both classes.

Insert, Replace, and Delete Operations

The insert method places new text at a specified index, shifting existing characters. The replace method changes a substring between given indexes. The delete method removes characters between specified indexes, altering the original string directly.

Other Useful String Methods

You can get the length of a string, reverse it, or print a substring of a specific length. Methods like charAt and lastIndexOf help access or search for characters within the string.

Switching Between StringBuffer and StringBuilder

The only difference in using StringBuffer or StringBuilder in code is the object creation; their methods and usage are otherwise the same. StringBuilder is generally preferred for single-threaded tasks due to better performance.

Synchronization vs. Non-Synchronization

Synchronization means only one thread can access a method at a time, preventing data corruption when multiple threads modify the same object. StringBuffer is synchronized, so it's safe for multithreaded use, but slower. StringBuilder is not synchronized, so it's faster but unsafe for concurrent use.

Understanding Threads and Processes

In Java, a process runs the code, and you can create multiple sub-processes called threads. By default, one thread runs the code, but you can create more for multitasking (multithreading). Threads can run simultaneously and may access shared resources.

Synchronized vs. Non-Synchronized Environments

In a non-synchronized environment, multiple threads can access and modify the same resource at any time, possibly causing conflicts and data corruption. Synchronization ensures only one thread modifies the resource at a time, maintaining data integrity.

Synchronization in StringBuffer

When a method like delete is used on a StringBuffer by one thread, other threads must wait until it finishes. This prevents other threads from making changes at the same time, avoiding errors and data loss.

When to Use StringBuffer or StringBuilder

StringBuffer should be used in multi-threaded environments to avoid data problems, while StringBuilder is suitable for single-threaded environments for better performance. Using

StringBuilder in multi-threading can lead to inconsistent or corrupted data.

Recap and Next Steps

The session summarized string operations, arrays, classes, and encapsulation. The next topics will include inheritance, and a test covering theory and programming questions will be held soon.