**Assignment Questions 2**

<aside> 💡 **Q1. What are the Conditional Operators in Java?**

</aside>

<aside> 💡 **Q2. What are the types of operators based on the number of operands?**

</aside>

<aside> 💡 **Q3.What is the use of Switch case in Java programming?**

</aside>

<aside> 💡 **Q4.What are the conditional Statements and use of conditional statements in Java?**

</aside>

<aside> 💡 **Q5.What is the syntax of if else statement?**

</aside>

<aside> 💡 **Q6.How do you compare two strings in Java?**

</aside>

<aside> 💡 **Q7.What is Mutable String in Java Explain with an example**

</aside>

<aside> 💡 **Q8.Write a program to sort a String Alphabetically**

</aside>

<aside> 💡 **Q9.Write a program to check if the letter 'e' is present in the word**

**'Umbrella'.**

</aside>

<aside> 💡 **Q10.Where exactly is the string constant pool located in the**

**memory?**

</aside>

**ANSWERS**

Q1. What are the Conditional Operators in Java?

Answer: In Java, conditional operators are used to perform logical operations and make decisions based on conditions. The conditional operators in Java are:

1. Conditional-AND (&&): This operator returns true if both operands are true; otherwise, it returns false. It evaluates the second operand only if the first operand is true.
2. Conditional-OR (||): This operator returns true if at least one of the operands is true; otherwise, it returns false. It evaluates the second operand only if the first operand is false.
3. Conditional-NOT (!): This operator is a unary operator that negates the boolean value of its operand. It returns true if the operand is false, and false if the operand is true.

These operators are primarily used in conditional statements, such as if-else statements and loops, to control the flow of the program based on specified conditions.

Q2. What are the types of operators based on the number of operands?

Answer: Operators in Java can be classified based on the number of operands they take. There are three types of operators:

1. Unary Operators: Unary operators work with a single operand. They perform various operations, such as negation, increment, decrement, and logical complement. Examples of unary operators include the unary minus (-), unary plus (+), increment (++), decrement (--), and logical complement (!) operators.
2. Binary Operators: Binary operators work with two operands. They perform operations like arithmetic operations (addition, subtraction, multiplication, division), comparison (greater than, less than), assignment, logical operations (AND, OR), and more. Examples of binary operators include the arithmetic operators (+, -, \*, /), relational operators (>, <, >=, <=), assignment operator (=), and logical operators (&&, ||).
3. Ternary Operator: The ternary operator (?:) is the only operator in Java that takes three operands. It is a shorthand way of writing an if-else statement. It evaluates a condition and returns one of two expressions based on whether the condition is true or false.

Q3. What is the use of Switch case in Java programming?

Answer: The switch-case statement in Java is used to control the flow of the program based on multiple possible values of a variable or an expression. It provides a convenient way to write multi-branch conditional statements.

The switch-case statement consists of multiple cases, each representing a specific value or range of values. The variable or expression being tested is compared with each case, and if a match is found, the corresponding block of code is executed. If no match is found, an optional default case can be provided to handle unmatched values.

The use of switch-case statement can simplify code and improve readability, especially when dealing with multiple conditions. It is often used as an alternative to long if-else if chains.

Q4. What are the conditional Statements and use of conditional statements in Java?

Answer: Conditional statements in Java are used to control the flow of execution based on certain conditions. They allow the program to make decisions and perform different actions depending on whether a condition is true or false.

The two main types of conditional statements in Java are:

1. If-else Statements: The if-else statement evaluates a condition and executes a block of code if the condition is true. If the condition is false, an optional else block is executed. This allows for two different paths of execution based on a single condition.
2. Switch-case Statements: The switch-case statement evaluates a variable or expression and matches it with different cases. It executes the code block associated with the first matching case. It provides an alternative to multiple if-else if statements when testing for different values of a variable or expression.

Conditional statements are fundamental in programming as they allow for decision-making and enable the program to respond to different scenarios. They help in controlling the flow of execution and executing specific code based on specified conditions.

Q5. What is the syntax of if-else statement?

Answer: The syntax of the if-else statement in Java is as follows:

javaCopy code

if (condition) { // code to be executed if the condition is true } else { // code to be executed if the condition is false }

The **if** statement is followed by a condition enclosed in parentheses. If the condition evaluates to **true**, the block of code within the if statement is executed. If the condition evaluates to **false**, the block of code within the else statement (optional) is executed.

The if-else statement allows for two possible paths of execution based on the outcome of the condition. If the condition is true, the code within the if block is executed, and if the condition is false, the code within the else block is executed.

Q6. How do you compare two strings in Java?

Answer: In Java, you can compare two strings using the **equals()** method or the **compareTo()** method. Here's how each method works:

1. **equals()** method: The **equals()** method is used to compare the content of two strings. It returns **true** if the content of the two strings is equal, and **false** otherwise. Here's an example:

javaCopy code

String str1 = "Hello"; String str2 = "World"; if (str1.equals(str2)) { System.out.println("The strings are equal."); } else { System.out.println("The strings are not equal."); }

Output:

sqlCopy code

The strings are not equal.

1. **compareTo()** method: The **compareTo()** method is used to perform lexicographic comparison of two strings. It returns an integer value indicating the relationship between the strings. If the first string is lexicographically less than the second string, it returns a negative value. If the strings are equal, it returns 0. If the first string is lexicographically greater than the second string, it returns a positive value. Here's an example:

javaCopy code

String str1 = "Apple"; String str2 = "Banana"; int result = str1.compareTo(str2); if (result < 0) { System.out.println("str1 is less than str2."); } else if (result == 0) { System.out.println("str1 is equal to str2."); } else { System.out.println("str1 is greater than str2."); }

Output:

csharpCopy code

str1 is less than str2.

Q7. What is Mutable String in Java? Explain with an example.

Answer: In Java, a mutable string refers to a string object whose value can be modified after it is created. The **String** class in Java is immutable, which means once a **String** object is created, its value cannot be changed.

However, there are mutable alternatives to **String** in Java, such as the **StringBuilder** and **StringBuffer** classes. These classes provide methods to modify the string content without creating a new object.

Here's an example that demonstrates the mutability of **StringBuilder**:

javaCopy code

StringBuilder sb = new StringBuilder("Hello"); System.out.println("Before modification: " + sb); // Output: Hello sb.append(" World"); System.out.println("After modification: " + sb); // Output: Hello World sb.insert(5, ","); // Insert a comma at index 5 System.out.println("After insertion: " + sb); // Output: Hello, World sb.delete(5, 7); // Delete characters fromindex 5 to 7 System.out.println("After deletion: " + sb); // Output: Hello World In this example, the `StringBuilder` object `sb` is initially created with the value "Hello". Using various methods like `append()`, `insert()`, and `delete()`, we modify the content of `sb` without creating a new object. This demonstrates the mutability of `StringBuilder`. Mutable strings are useful when we need to perform multiple string modifications efficiently, as creating a new string for each modification can be inefficient in terms of memory and performance. Q8. Write a program to sort a String Alphabetically. Here's an example program that sorts a string alphabetically in Java: ```java import java.util.Arrays; public class StringSortExample { public static void main(String[] args) { String str = "openai"; char[] charArray = str.toCharArray(); Arrays.sort(charArray); String sortedStr = new String(charArray); System.out.println("Original String: " + str); System.out.println("Sorted String: " + sortedStr); } }

Output:

arduinoCopy code

Original String: openai Sorted String: aeinop

In this program, we first convert the string **str** into a character array using the **toCharArray()** method. Then, we use the **Arrays.sort()** method to sort the character array in ascending order. Finally, we create a new string **sortedStr** from the sorted character array using the **String** class constructor, and print the original and sorted strings.

Q9. Write a program to check if the letter 'e' is present in the word 'Umbrella'.

Here's an example program to check if the letter 'e' is present in the word 'Umbrella':

javaCopy code

public class LetterCheckExample { public static void main(String[] args) { String word = "Umbrella"; if (word.contains("e")) { System.out.println("The letter 'e' is present in the word."); } else { System.out.println("The letter 'e' is not present in the word."); } } }

Output:

arduinoCopy code

The letter 'e' is present in the word.

In this program, we use the **contains()** method of the **String** class to check if the letter 'e' is present in the string **word**. If the **contains()** method returns **true**, it means the letter 'e' is present in the word. Otherwise, it is not present.

Q10. Where exactly is the string constant pool located in the memory?

The string constant pool in Java is located in the heap memory area. More specifically, it is a part of the runtime constant pool, which is a section of the heap dedicated to storing constant values, such as string literals and symbolic references used by the code.

The string constant pool is a special area within the runtime constant pool that is reserved for storing unique string literals. When a string literal is encountered in the code, Java checks if an identical string already exists in the pool. If it does, the reference to the existing string is returned. If not, a new string object is created and added to the pool, and its reference is returned.

The string constant pool being located in the heap memory allows it to be garbage collected like other objects. When there are no more references to a string in the pool, it becomes eligible for garbage collection and its memory is reclaimed.

Note that starting from Java 7, some parts of the string constant pool can be stored in the native memory outside the heap for performance optimization. However, the conceptual understanding remains the same, with the string constant pool being part of the overall heap memory area.