**Assignment Questions 2**

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

In Java, there are three conditional operators:  
  
1. Ternary Operator (?): It is also known as the conditional operator. It is used to evaluate a condition and return one of two values based on the result. The syntax is "condition ? value1 : value2". If the condition is true, it returns value1; otherwise, it returns value2.  
  
2. Equality Operators (== and !=): The equality operators are used to compare two values for equality. The "==" operator checks if two values are equal, while the "!=" operator checks if they are not equal. These operators can be used with numeric types, characters, and object references.  
  
3. Relational Operators (<, >, <=, >=): The relational operators are used to compare the relationship between two values. They return a boolean value indicating whether the comparison is true or false. The operators include less than (<), greater than (>), less than or equal to (<=), and greater than or equal to (>=).  
  
These conditional operators in Java allow for making decisions and performing comparisons based on conditions, enabling control flow and logic in programming.

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

Operators in Java can be categorized based on the number of operands they operate on:  
  
1. Unary Operators: These operators work on a single operand. Examples include the unary plus (+), unary minus (-), increment (++), decrement (--), logical complement (!), and bitwise complement (~) operators.  
  
2. Binary Operators: Binary operators work on two operands. They include arithmetic operators (+, -, \*, /, %), relational operators (>, <, >=, <=, ==, !=), logical operators (&&, ||), bitwise operators (&, |, ^), assignment operators (=, +=, -=, \*=, /=), and more.  
  
3. Ternary Operator: The ternary operator (?:) is the only operator in Java that takes three operands. It evaluates a condition and returns one of two possible values based on the result of the condition.  
  
These different types of operators allow programmers to perform various operations and computations in Java, providing flexibility and control over program behavior.

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

The switch case statement in Java is used for efficient branching and decision-making based on the value of an expression. Here are the main uses of the switch case:  
  
1. Simplify Multiple Conditionals: Switch case simplifies code readability when dealing with multiple conditional statements. It provides a concise way to check different values of an expression without writing a series of if-else statements.  
  
2. Selective Execution: Switch case allows selective execution of code blocks based on the matched value. It compares the expression with multiple cases and executes the block of code corresponding to the matching case. This helps in implementing different actions or behaviors based on different input values.  
  
3. Code Organization: Switch case improves code organization by grouping related cases together. It provides a clear structure for handling different cases, making the code easier to understand and maintain.  
  
4. Performance Optimization: Switch case is optimized for performance by using jump tables or lookup tables behind the scenes. This results in efficient execution, especially when dealing with a large number of cases.

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

Conditional statements in Java are used to control the flow of execution based on certain conditions. The main conditional statements in Java are if, if-else, if-else if ladder, and switch case.

These statements allow the program to make decisions and execute different blocks of code based on the evaluation of a condition. Conditional statements help in implementing logic and handling various scenarios dynamically. They enable the program to respond differently to different inputs or conditions, making the code more flexible and adaptable. By using conditional statements, programmers can control the execution flow, handle user input, implement branching logic, and create decision-making structures in Java programs.

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

- The `if` keyword is used to begin the if-else statement.

- The condition to be evaluated is placed within parentheses `()`.

- If the condition evaluates to true, the code block within the first set of curly braces `{}` is executed.

- If the condition is false, the code block within the second set of curly braces `{}` (the `else` block) is executed.

- The `else` keyword is used to specify the alternative code block to be executed if the condition is false.

- The `else` block is optional, and it can be omitted if no action is required for the false condition.

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

1. Using the `equals()` method: The `equals()` method compares the content of two strings for equality. It returns `true` if the strings have the same characters in the same order, and `false` otherwise. Here's an example:

String str1 = "hello";

String str2 = "HELLO";

if (str1.equalsIgnoreCase(str2)) {

// Strings are equal (ignoring case)

} else {

// Strings are not equal

}

1. Using the `equalsIgnoreCase()` method: The `equalsIgnoreCase()` method compares two strings while ignoring their case. It returns `true` if the strings have the same characters, regardless of their case. Here's an example:

String str1 = "hello";

String str2 = "HELLO";

if (str1.equalsIgnoreCase(str2)) {

// Strings are equal (ignoring case)

} else {

// Strings are not equal

}

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

In Java, a mutable string refers to a string object that can be modified or changed after its creation. The `StringBuilder` class provides mutable string functionality, allowing efficient concatenation and modification of strings. Here's an example:

StringBuilder mutableString = new StringBuilder("Hello");

mutableString.append(" World"); // Modify the string by appending

mutableString.insert(5, " Java"); // Insert a substring at a specific position

mutableString.delete(0, 6); // Remove a portion of the string

String result = mutableString.toString(); // Convert back to an immutable string

💡 **Q8.Write a program to sort a String Alphabetically**

import java.util.Arrays;

public class StringSortExample {

public static void main(String[] args) {

String str = "openai";

// Convert the string to a character array

char[] charArray = str.toCharArray();

// Sort the character array

Arrays.sort(charArray);

// Create a new string from the sorted character array

String sortedStr = new String(charArray);

System.out.println("Original string: " + str);

System.out.println("Sorted string: " + sortedStr);

}

}

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

**'Umbrella'.**

public class LetterCheckExample {

public static void main(String[] args) {

String word = "Umbrella";

char target = 'e';

boolean isPresent = false;

for (int i = 0; i < word.length(); i++) {

if (word.charAt(i) == target) {

isPresent = true;

break;

}

}

if (isPresent) {

System.out.println("The letter '" + target + "' is present in the word '" + word + "'.");

} else {

System.out.println("The letter '" + target + "' is not present in the word '" + word + "'.");

}

}

}

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

**memory?**

The string constant pool in Java is located in the PermGen (Permanent Generation) space, which is a part of the Java Virtual Machine (JVM) memory. However, starting from Java 8, with the introduction of the Metaspace memory area, the string constant pool is now stored in the Metaspace instead of the PermGen space. The string constant pool is a special area in memory where Java stores unique string literals to optimize memory usage by reusing the same instance of a string if it already exists, reducing memory overhead.