**Assignment Questions 2**

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

The conditional operator `? :` in Java is also known as ternary operator used for making decisions based on a Boolean expression. It evaluates the Boolean expression and returns one of two expressions depending on its truth value.

The syntax is `booleanExpression ? expression1 : expression2`. If the `booleanExpression` is true, `expression1` is returned; otherwise, `expression2` is returned. It allows for compact conditional assignments or expressions. However, it should be used judiciously to maintain code readability.

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

Based on the number of operands they operate on, operators in Java can be classified into three categories:

1. Unary Operators:

- Unary operators operate on a single operand.

- Examples: `+` (unary plus), `-` (unary minus), `++` (increment), `--` (decrement), `!` (logical complement), etc.

2. Binary Operators:

- Binary operators operate on two operands.

- Examples: `+` (addition), `-` (subtraction), `\*` (multiplication), `/` (division), `%` (remainder/modulo), `==` (equality), `!=` (inequality), `>` (greater than), `<` (less than), `&&` (logical AND), `||` (logical OR), etc.

3. Ternary Operator:

- The ternary operator is the only operator in this category.

- It takes three operands and is denoted by `? :`.

- It evaluates a Boolean expression and returns one of two expressions based on the result.

- Example: `booleanExpression ? expression1 : expression2`

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

The switch statement in Java is a control flow statement used for selective execution based on the value of an expression. It provides a convenient way to handle multiple possible cases or conditions in a concise and structured manner. The switch statement is primarily used for the following purposes:

1. Multiple Branching: The switch statement allows the program to select one of many possible branches of code execution based on the value of an expression. It provides an alternative to using multiple nested if-else statements.

2. Code Organization: Switch statements improve code organization and readability when dealing with a large number of mutually exclusive cases. Instead of writing separate if-else if-else blocks for each case, a switch statement provides a more compact and structured approach.

3. Efficiency: Switch statements can be more efficient than multiple if-else statements in certain cases. The JVM can optimize the switch statement by using a jump table or binary search, resulting in faster execution compared to linearly evaluating multiple if-else conditions.

Here's the basic syntax of a switch statement in Java:

switch (expression) {

case value1:

// Code to be executed when expression matches value1

break;

case value2:

// Code to be executed when expression matches value2

break;

// ...

default:

// Code to be executed when expression does not match any case

}

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

Conditional statements in Java are control flow statements that allow the execution of different code blocks based on certain conditions. They enable the program to make decisions and perform actions accordingly. Java provides several conditional statements:

1. if Statement:

- The `if` statement allows the execution of a block of code if a specified condition is true.

- It can be followed by an optional `else` statement to specify an alternative block of code to execute if the condition is false.

2. if-else Statement:

- The `if-else` statement extends the `if` statement by providing an alternative block of code to execute when the condition is false.

- If the condition in the `if` statement is true, the code within the `if` block is executed. Otherwise, the code within the `else` block is executed.

3. Nested if-else Statement:

- The `if-else if-else` statement allows multiple conditions to be tested sequentially.

- It provides a way to check for multiple cases and execute different code blocks based on the first matching condition.

4. Switch Statement:

- The `switch` statement provides a multi-branching mechanism to execute different code blocks based on the value of an expression.

- It evaluates the expression and matches it against a series of `case` values. When a match is found, the corresponding code block is executed.

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

if(condition){

//block to execute when the condition is true

}

else{

//this block will execute when the condition is false

}

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

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 for equality.

- It returns `true` if the content of the strings is the same, and `false` otherwise.

- The comparison is case-sensitive, so uppercase and lowercase characters are considered different.

- Syntax: `string1.equals(string2)`

- Example:

```java

String str1 = "Hello";

String str2 = "hello";

System.out.println(str1.equals(str2)); // Output: false

```

2. `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.

- It follows the Unicode value of the characters to determine the order.

- If the strings are equal, it returns 0. If the first string is lexicographically greater, it returns a positive value. If the first string is lexicographically smaller, it returns a negative value.

- Syntax: `string1.compareTo(string2)`

- Example:

```java

String str1 = "apple";

String str2 = "banana";

int result = str1.compareTo(str2);

if (result < 0) {

System.out.println("str1 comes before str2");

} else if (result > 0) {

System.out.println("str1 comes after str2");

} else {

System.out.println("str1 and str2 are equal");

}

```

Output: "str1 comes before str2"

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

In Java, strings are immutable, meaning their values cannot be changed once they are created. However, there are mutable alternatives to strings, such as StringBuilder and StringBuffer, which can be modified after creation. StringBuilder is a class in Java that provides a mutable sequence of characters. It allows you to modify the contents of the string without creating a new string object each time you make changes.

Unlike regular strings, where concatenation results in a new string object, StringBuilder allows you to efficiently modify and build up strings without excessive memory overhead. It provides methods for appending, inserting, deleting, and replacing characters within the mutable sequence.

It's worth noting that StringBuilder is not thread-safe, meaning it is not suitable for use in multi-threaded environments. If thread-safety is required, the StringBuffer class can be used instead, which provides similar functionality but with thread-safe operations.

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

public class StringSortExample {

public static void main(String[] args) {

String input = "openai";

// Convert the string to a character array

char[] charArray = input.toCharArray();

// Perform Bubble Sort

int length = charArray.length;

for (int i = 0; i < length - 1; i++) {

for (int j = 0; j < length - i - 1; j++) {

if (charArray[j] > charArray[j + 1]) {

// Swap the characters

char temp = charArray[j];

charArray[j] = charArray[j + 1];

charArray[j + 1] = temp;

}

}

}

// Create a new string from the sorted character array

String sortedString = new String(charArray);

// Print the sorted string

System.out.println("Sorted String: " + sortedString);

}

}

**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";

boolean isPresent = false;

// Iterate through each character in the word

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

// Check if the current character is 'e'

if (word.charAt(i) == 'e') {

isPresent = true;

break;

}

}

// Print the result

if (isPresent) {

System.out.println("'e' is present in the word.");

} else {

System.out.println("'e' is not present in the word.");

}

}

}

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

In Java, the string constant pool is a part of the Java heap memory. The Java heap is the runtime data area where objects are allocated, including strings. The string constant pool, specifically, is a special area within the Java heap that is used to store string literals, which are string constants defined in the source code.