Assignment Questions 2

Q1. What are the Conditional Operators in Java?

In Java, conditional operators are used to perform conditional (or ternary) operations, allowing you to make decisions based on certain conditions. The conditional operator is denoted by the question mark ? and the colon : and has the following syntax:

Syntax:

condition ? expression1 : expression2

Here, condition is a boolean expression that evaluates to true or false. If the condition is true, then expression1 is executed; otherwise, expression2 is executed. The result of the entire conditional operation is the value of either expression1 or expression2, depending on the evaluation of the condition.

Example:

int num = 10;

String result = (num > 5) ? "Greater than 5" : "Less than or equal to 5";

System.out.println(result); // Output: "Greater than 5"

In this example, the condition num > 5 evaluates to true, so the value of expression1 ("Greater than 5") is assigned to the variable result.

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

Operators in programming languages can be categorized based on the number of operands they require. The three main types of operators based on the number of operands are:

Unary Operators:

Unary operators work with a single operand. They perform operations on the operand and typically have a prefix or postfix notation. Common examples of unary operators include:

Increment (++): Increases the value of the operand by 1.

Decrement (--): Decreases the value of the operand by 1.

Logical NOT (!): Inverts the boolean value of the operand (true becomes false and vice versa).

Unary Plus (+): Represents the identity of the operand (e.g., +x is the same as x).

Unary Minus (-): Negates the value of the operand.

Binary Operators:

Binary operators work with two operands and are the most common type of operators. They perform operations between the two operands and have infix notation. Common examples of binary operators include:

Addition (+): Adds two operands.

Subtraction (-): Subtracts the second operand from the first operand.

Multiplication (*): Multiplies two operands.

Division (/): Divides the first operand by the second operand.

Modulus (%): Calculates the remainder of the division of the first operand by the second operand.

Ternary Operator:

The ternary operator (conditional operator) is unique as it takes three operands. It is used for conditional expressions and returns one of two values based on a given condition. The syntax of the ternary operator is:

condition ? expression1 : expression2

If the condition is true, it evaluates expression1; otherwise, it evaluates expression2.

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

The switch case statement in Java is used to execute different blocks of code based on the value of an expression. It provides an efficient and organized way to handle multiple possible cases for a single variable or expression. The switch case statement is particularly useful when dealing with multiple if-else conditions.

The syntax of the switch case statement is as follows:

```
switch (expression) {
    case value1:
        // Code to execute if expression is equal to value1
        break;
    case value2:
        // Code to execute if expression is equal to value2
        break;
    // Additional cases...
    default:
        // Code to execute if expression doesn't match any case
        break;
}
```

Here's how the switch case statement works:

The expression is evaluated.

The value of the expression is compared with the values in each case.

If a match is found, the corresponding block of code for that case is executed.

The break statement is used to exit the switch case block after executing the code for a matched case. Without the break, the code execution will continue to the next case until a break is encountered or the end of the switch case block is reached.

If no match is found, the code within the default block (optional) is executed. The default block is used when none of the cases match the value of the expression.

The switch case statement provides a more concise and readable way to handle multiple conditions compared to using nested if-else statements, especially when the conditions are based on constant values. However, it's important to note that the switch case statement can only be used with primitive data types, enumerated types, and String objects in Java 7 and later versions.

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

Conditional statements in Java allow you to control the flow of execution based on certain conditions. They help you make decisions and execute specific blocks of code based on whether a condition is true or false. There are three main types of conditional statements in Java: if statement, if-else statement, and switch case statement.

If Statement:

The if statement allows you to execute a block of code if a given condition is true. It has the following syntax:

java

Copy code

```
if (condition) {
    // Code to execute if the condition is true
}
```

Example:

```
int age = 25;
if (age >= 18) {
    System.out.println("You are an adult.");
}
```

If-Else Statement:

The if-else statement allows you to execute one block of code if the condition is true, and another block of code if the condition is false. It has the following syntax:

java

Copy code

```
if (condition) {
    // Code to execute if the condition is true
} else {
    // Code to execute if the condition is false
```

}

Example:

```
int age = 15;

if (age >= 18) {

    System.out.println("You are an adult.");

} else {

    System.out.println("You are a minor.");

}
```

Switch Case Statement:

The switch case statement is used to select one of many code blocks to be executed. It compares the value of an expression against multiple cases and executes the block of code associated with the matching case. It has the following syntax:

java

Copy code

```
switch (expression) {

    case value1:

        // Code to execute if expression is equal to value1

        break;

    case value2:

        // Code to execute if expression is equal to value2

        break;

    // Additional cases...

    default:

        // Code to execute if expression doesn't match any case
```

```
        break;
}
```

Example:

```
int dayOfWeek = 3;
switch (dayOfWeek) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    // Additional cases...
    default:
        System.out.println("Invalid day");
        break;
}
```

Conditional statements are essential for making decisions in programs and allow you to control the program's flow based on various conditions, making your code more flexible and efficient.

Q5.What is the syntax of if else statement?

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

```
if (condition) {

   // Code to execute if the condition is true

} else {

   // Code to execute if the condition is false

}
```

Here's an explanation of each part of the syntax:

if: This keyword is used to start the if statement.

condition: This is a boolean expression that evaluates to either true or false. If the condition is true, the code inside the if block will be executed. If the condition is false, the code inside the else block (if present) will be executed.

{} (curly braces): These are used to enclose the blocks of code associated with the if and else statements. The code inside the curly braces will be executed if the corresponding condition is true or false.

else: This keyword is optional. It is used to specify an alternative block of code that should be executed when the condition in the if statement is false.

The if-else statement provides a way to perform two different sets of actions based on whether a given condition is true or false. If the condition is true, the code inside the if block will be executed. If the condition is false, the code inside the else block (if present) will be executed. The else block is optional; if it is not included, nothing will be executed 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. The appropriate method to use depends on the comparison context.

Using 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 the same and false otherwise.

java

Copy code

```
String str1 = "Hello";

String str2 = "hello";
```

```
if (str1.equals(str2)) {
    System.out.println("str1 and str2 are equal.");
} else {
    System.out.println("str1 and str2 are not equal.");
}
```

Output:

str1 and str2 are not equal.

Using compareTo() method:

The compareTo() method is used to perform lexicographic comparison of two strings. It returns an integer value indicating whether the first string is lexicographically greater than, equal to, or less than the second string.

```
String str1 = "apple";
String str2 = "banana";

int result = str1.compareTo(str2);

if (result == 0) {
    System.out.println("str1 and str2 are equal.");
} else if (result < 0) {
    System.out.println("str1 comes before str2.");
} else {
    System.out.println("str1 comes after str2.");
}
```

Output:

str1 comes before str2.

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

In Java, a mutable string refers to a string that can be modified or changed after it is created. In contrast, a mutable string can be altered, and its content can be modified during the program execution.

In Java, the StringBuilder class provides a mutable string that can be modified efficiently. It is part of the Java Standard Library and is commonly used when you need to perform multiple string manipulations or concatenations.

Here's an example to demonstrate the concept of a mutable string using StringBuilder:

```
public class MutableStringExample {

    public static void main(String[] args) {

        // Creating a StringBuilder object

        StringBuilder stringBuilder = new StringBuilder("Hello");



        // Appending additional text to the mutable string

        stringBuilder.append(", ");

        stringBuilder.append("Java ");

        // Inserting text at a specific position in the string

        stringBuilder.insert(6, "World ");



        // Replacing text in the mutable string

        stringBuilder.replace(0, 5, "Greetings");



        // Deleting characters from the mutable string

        stringBuilder.delete(11, 17);
```

```
        // Converting the mutable string to an immutable String object
        String finalString = stringBuilder.toString();

        // Printing the final modified string
        System.out.println(finalString);
    }
}
```

Output:

Greetings, World

In this example, we first create a StringBuilder object with the initial value "Hello." Then, we perform various string operations on the StringBuilder, such as appending, inserting, replacing, and deleting characters. Since StringBuilder is mutable, these operations can be performed efficiently without creating new objects.

Finally, when we have finished modifying the string, we convert it back to an immutable String object using the toString() method of StringBuilder. The final output demonstrates the result of all the string modifications.

Q8.Write a program to sort a String Alphabetically

To sort a string alphabetically in Java, you can convert the string to a character array, sort the array using the Arrays.sort() method, and then create a new string from the sorted character array. Here's an example program:

```
import java.util.Arrays;

public class StringSortAlphabetically {
    public static void main(String[] args) {
        String inputString = "hello world";
        String sortedString = sortStringAlphabetically(inputString);
```

```
        System.out.println("Original String: " + inputString);
        System.out.println("Sorted String: " + sortedString);
    }

    public static String sortStringAlphabetically(String inputString) {
        // Convert the input string to a character array
        char[] charArray = inputString.toCharArray();

        // Sort the character array in ascending order
        Arrays.sort(charArray);

        // Create a new string from the sorted character array
        return new String(charArray);
    }
}
```

In this example, we define a method sortStringAlphabetically that takes the input string and returns the sorted string. Inside the method, we convert the input string to a character array, sort the array using Arrays.sort(), and then create a new string from the sorted character array.

Note that the sorting is case-sensitive, so uppercase letters will come before lowercase letters in the sorted string. If you want a case-insensitive sorting, you can convert the string to lowercase or uppercase before sorting.

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

'Umbrella'.

```
public class LetterCheck {
```

```
    public static void main(String[] args) {
        String word = "Umbrella";
        char letterToCheck = 'e';

        boolean isPresent = checkLetter(word, letterToCheck);

        if (isPresent) {
            System.out.println("The letter '" + letterToCheck + "' is present in the word '" + word + "'.");
        } else {
            System.out.println("The letter '" + letterToCheck + "' is not present in the word '" + word + "'.");
        }
    }

    public static boolean checkLetter(String word, char letter) {
        for (char c : word.toCharArray()) {
            if (c == letter) {
                return true;
            }
        }
        return false;
    }
}
```

Output:

The letter 'e' is present in the word 'Umbrella'.

In this program, we define a method checkLetter that takes a word and a character as input and returns true if the character is present in the word, and false otherwise. We then call this method with the word "Umbrella" and the letter 'e' to check if 'e' is present in the word. The result is then printed to the console.

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

In Java, the string constant pool is a part of the runtime constant pool, which is a specific area in the Java Virtual Machine (JVM) memory. The runtime constant pool is a pool of constants that are used during the execution of a Java program.

The string constant pool is a pool of unique string literals created by the Java compiler. When you create a string using a string literal (e.g., "Hello"), the JVM checks if the string with the same value already exists in the string constant pool. If it does, the JVM returns a reference to the existing string; otherwise, it creates a new string in the pool.

The exact location of the runtime constant pool in memory may vary depending on the JVM implementation, but it is typically located in the method area of the JVM memory. The method area is a part of the JVM memory used to store class-level information, including bytecode, field information, and method information.

It's important to note that the string constant pool is a part of the memory management mechanism in the JVM, and its implementation details may vary across different JVM implementations. As a Java developer, you don't need to worry about the exact memory location; the JVM takes care of managing the string constant pool for you.