JAVA ASSIGNMENT 2

Q1. What are the Conditional Operators in Java?

ANSWER:-

In Java, conditional operators are used to perform logical or relational comparisons and return a boolean value based on the result. There are three conditional operators in Java:

1. Conditional AND (&&): The conditional AND operator evaluates two boolean expressions and returns true if both expressions are true. If the first expression is false, the second expression is not evaluated because the overall result would already be false.
For example:

```
boolean result = (x > 5) && (y < 10);
```

In this example, the variable "result" will be true if the value of "x" is greater than 5 and the value of "y" is less than 10.

2. Conditional OR (||): The conditional OR operator evaluates two boolean expressions and returns true if either of the expressions is true. If the first expression is true, the second expression is not evaluated because the overall result would already be true.
For example:

```
boolean result = (x > 5) || (y < 10);
```

In this example, the variable "result" will be true if the value of "x" is greater than 5 or the value of "y" is less than 10.

3. Conditional NOT (!): The conditional NOT operator reverses the logical state of its operand. If the operand is true, the NOT operator returns false, and if the operand is false, it returns true.
For example:

```
boolean result = !(x > 5);
```

In this example, the variable "result" will be true if the value of "x" is not greater than 5.

These conditional operators are commonly used in decision-making constructs like if-else statements and loops to control the flow of execution based on certain conditions.

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

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

1. Unary Operators: Unary operators act on a single operand. They perform operations on a single value or variable. Java provides several unary operators:

- Unary plus (+): Represents the identity operation. It simply returns the value of the operand.
- Unary minus (-): Negates the value of the operand.
- Increment (++) and Decrement (--): Used to increase or decrease the value of the operand by 1.
- Logical complement (!): Reverses the logical state of the operand.
- Bitwise complement (~): Inverts the bits of the operand.

Example:

```
int x = 5;
int result = -x;
System.out.println(result); // Output: -5
```

2. Binary Operators: Binary operators work on two operands. They perform operations on two values or variables and return a result. Java supports various binary operators:

- Arithmetic Operators: Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulo (%).
- Relational Operators: Equality (==), Inequality (!=), Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=).
- Logical Operators: Conditional AND (&&), Conditional OR (||).
- Assignment Operators: Assigns a value to a variable. Examples: =, +=, -=, *=, /=.
- Bitwise Operators: Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^), Left shift (<<), Right shift (>>).

Example:

```
int x = 5;
int y = 10;
int sum = x + y;
System.out.println(sum); // Output: 15
```

3. Ternary Operator: The ternary operator is the only operator in Java that takes three operands. It is a shorthand version of an if-else statement. The syntax is as follows: `(condition) ? expression1 : expression2`. If the condition is true, expression1 is evaluated; otherwise, expression2 is evaluated.

Example:

```
int x = 5;
int y = 10;
int max = (x > y) ? x : y;
```

```
System.out.println(max); // Output: 10
```

These are the different types of operators based on the number of operands they work with in Java.

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

ANSWER:-

The switch-case statement in Java is a control flow statement that allows you to execute different blocks of code based on the value of a variable or expression. It provides an alternative to using multiple if-else-if statements when you have a single variable to compare against multiple values.

The switch-case statement consists of a variable or expression called the "switch" and a series of "case" labels. The switch statement evaluates the value of the switch expression and compares it with the values specified in the case labels. When a match is found, the corresponding block of code associated with that case label is executed. If no match is found, an optional "default" case can be used to specify a block of code that executes when none of the other cases match.

The switch-case statement can help improve the readability and maintainability of your code in situations where you have a fixed set of possible values to compare against. It is particularly useful when dealing with enum types or integer values.

Here's a basic example to illustrate the use of switch-case in Java:

```
int day = 3;
String dayOfWeek;

switch (day) {
   case 1:
      dayOfWeek = "Monday";
      break;
   case 2:
      dayOfWeek = "Tuesday";
      break;
   case 3:
      dayOfWeek = "Wednesday";
      break;
   case 4:
      dayOfWeek = "Thursday";
      break;
   case 5:
```

```
        dayOfWeek = "Friday";
        break;
    case 6:
        dayOfWeek = "Saturday";
        break;
    case 7:
        dayOfWeek = "Sunday";
        break;
    default:
        dayOfWeek = "Invalid day";
        break;
}

System.out.println(dayOfWeek); // Output: Wednesday
```

In this example, the value of the variable "day" is evaluated, and the corresponding case label is matched. Since "day" is 3, the code block associated with `case 3` is executed, and the value of "dayOfWeek" is set to "Wednesday". Without the switch-case statement, you would need to use multiple if-else statements to achieve the same result, which could become more cumbersome and less readable as the number of cases increases.

It's important to note that after executing a matching case block, the code will continue to execute the subsequent case blocks unless a "break" statement is encountered. The "break" statement is used to exit the switch statement and prevents fall-through to the next case.

Overall, the switch-case statement provides a concise and structured way to handle multiple possible values for a variable or expression in Java.

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

ANSWER:-

Conditional statements, also known as control flow statements, allow you to control the execution of code based on certain conditions. In Java, there are three main types of conditional statements:

1. if Statement:
The "if" statement is the most basic conditional statement. It allows you to execute a block of code if a given condition is true. If the condition is false, the code block is skipped. The syntax of the "if" statement is as follows:

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

Example:

```
int x = 5;
if (x > 0) {
    System.out.println("x is positive");
}
```

In this example, if the value of "x" is greater than 0, the message "x is positive" will be printed.

2. if-else Statement:
The "if-else" statement extends the "if" statement by providing an alternative code block to be executed when the condition is false. The syntax is as follows:

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

Example:

```
int x = 5;
if (x > 0) {
    System.out.println("x is positive");
} else {
    System.out.println("x is non-positive");
}
```

In this example, if the value of "x" is greater than 0, the message "x is positive" will be printed. Otherwise, the message "x is non-positive" will be printed.

3. if-else if-else Statement:
The "if-else if-else" statement allows you to check multiple conditions and execute different code blocks accordingly. It is useful when you have more than two possible outcomes. The syntax is as follows:

```
if (condition1) {
    // Code to be executed if condition1 is true
} else if (condition2) {
    // Code to be executed if condition2 is true
} else {
    // Code to be executed if all conditions are false
}
```

Example:

```
int x = 5;
if (x > 0) {
```

```
    System.out.println("x is positive");
} else if (x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is zero");
}
```

In this example, if the value of "x" is greater than 0, the message "x is positive" will be printed. If it is less than 0, the message "x is negative" will be printed. Otherwise, if the value is 0, the message "x is zero" will be printed.

Conditional statements are essential for making decisions and controlling the flow of your program based on different conditions. They allow you to write flexible and dynamic code that can respond to changing situations.

Q5.What is the syntax of if else statement?

ANSWER:-

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

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

Here's a breakdown of the syntax elements:

1. The keyword "if" is followed by an opening parenthesis "(".
2. Inside the parenthesis, you specify the condition that needs to be evaluated. The condition must be a boolean expression or a result that can be converted to a boolean value (true or false).
3. After the closing parenthesis ")", an opening curly brace "{" starts the code block that will be executed if the condition is true.
4. Inside the code block, you write the statements or code that will be executed if the condition is true.
5. The closing curly brace "}" marks the end of the if-block.
6. After the if-block, the keyword "else" is followed by another opening curly brace "{".
7. Inside the else-block, you write the statements or code that will be executed if the condition is false.
8. The closing curly brace "}" marks the end of the else-block.

The else-block is optional. If the condition in the if-statement evaluates to true, the if-block is executed. If the condition evaluates to false, the else-block is executed. In this way, the

if-else statement allows you to choose between two alternative paths of execution based on a condition.

Here's an example to illustrate the syntax of the if-else statement:

```
int number = 7;

if (number % 2 == 0) {
    System.out.println("The number is even.");
} else {
    System.out.println("The number is odd.");
}
```

In this example, the condition `number % 2 == 0` checks if the number is divisible by 2 without a remainder. If the condition is true, the message "The number is even." is printed. Otherwise, if the condition is false, the message "The number is odd." is printed.

It's important to note that the if-else statement can also be nested inside other if-else statements to create more complex decision-making structures.

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 you can use each method:

1. Using the `equals()` method:
The `equals()` method is used to compare the content of two strings for equality. It returns a boolean value (`true` or `false`) based on whether the strings have the same sequence of characters. Here's the syntax:

```
String str1 = "Hello";
String str2 = "World";

if (str1.equals(str2)) {
    // Strings are equal
} else {
    // Strings are not equal
}
```

In this example, the `equals()` method is called on `str1` and passed `str2` as an argument. If the two strings have the same content, the condition evaluates to `true`, and the code inside the `if` block is executed.

2. Using the `compareTo()` method:
The `compareTo()` method is used to compare strings lexicographically (based on the Unicode values of the characters). It returns an integer value that indicates the relationship between the strings. The value `0` indicates that the strings are equal, a negative value indicates that the calling string is lexicographically smaller, and a positive value indicates that the calling string is lexicographically larger. Here's the syntax:

```
String str1 = "Hello";
String str2 = "World";

int result = str1.compareTo(str2);

if (result == 0) {
    // Strings are equal
} else if (result < 0) {
    // str1 is lexicographically smaller than str2
} else {
    // str1 is lexicographically larger than str2
}
```

In this example, the `compareTo()` method is called on `str1` and passed `str2` as an argument. The result of the comparison is stored in the `result` variable. Depending on the value of `result`, the appropriate code block is executed to handle the comparison result.

It's important to note that when comparing strings, it's generally recommended to use the `equals()` method for content-based equality comparison. The `compareTo()` method is more suitable when you need to determine the lexicographic ordering of strings.

Additionally, if you want to perform a case-insensitive comparison, you can use the `equalsIgnoreCase()` method instead of `equals()` or apply the `compareToIgnoreCase()` method instead of `compareTo()`.

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

ANSWER:-

In Java, strings are immutable, which means that once a string object is created, its value cannot be changed. However, there is a mutable counterpart to the `String` class called `StringBuilder`.

The `StringBuilder` class provides a mutable sequence of characters that can be modified without creating a new object for each modification. This can be useful in scenarios where you need to perform a series of string manipulations efficiently.

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

```
StringBuilder mutableString = new StringBuilder("Hello");
mutableString.append(", World!");  // Appending additional text
mutableString.insert(5, " Java");  // Inserting text at a specific position

System.out.println(mutableString.toString());  // Output: Hello Java, World!
```

In this example, we create a `StringBuilder` object called `mutableString` and initialize it with the value "Hello". The `append()` method is then used to concatenate the string ", World!" to the existing value, modifying the original string. After that, the `insert()` method is used to insert the string " Java" at the 5th position in the `mutableString`.

Finally, the `toString()` method is called on the `mutableString` object to convert it back to a regular `String` so that we can print its value. The output will be "Hello Java, World!", demonstrating how the `StringBuilder` object allows for mutable string operations.

The benefit of using `StringBuilder` instead of regular string concatenation (`+` operator) or creating new `String` objects for each modification is that `StringBuilder` provides better performance and memory efficiency, especially when performing multiple string modifications.

It's important to note that if you are performing a small number of string manipulations or working with a limited number of strings, the immutability of `String` objects may not have a significant impact on performance or memory usage. However, when dealing with frequent and intensive string manipulations, using `StringBuilder` can provide noticeable performance improvements.

Q8.Write a program to sort a String Alphabetically

ANSWER:-

Certainly! Here's a Java program that sorts a string alphabetically using the `Arrays` class:

```
import java.util.Arrays;

public class StringSorter {
    public static void main(String[] args) {
        String input = "OpenAI";

        // Convert the string to an array of characters
        char[] charArray = input.toCharArray();

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

```
        // Convert the sorted array back to a string
        String sortedString = new String(charArray);

        // Print the sorted string
        System.out.println("Sorted String: " + sortedString);
    }
}
```

In this program, we take the input string "OpenAI" and convert it to a character array using the `toCharArray()` method. Then, we use the `Arrays.sort()` method to sort the character array in ascending order. Finally, we convert the sorted array back to a string using the `String` constructor, and the sorted string is printed.

The output of this program will be:

Sorted String: AIOeKnP

As you can see, the original string "OpenAI" is sorted alphabetically to "AIOeKnP".

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

ANSWER:-

Certainly! Here's a Java program that checks if the letter 'e' is present in the word 'Umbrella':

```
public class LetterChecker {

    public static void main(String[] args) {

        String word = "Umbrella";

        char targetLetter = 'e';

        boolean isPresent = false;


        // Iterate through each character in the word
```

```
        for (int i = 0; i < word.length(); i++) {
            // Check if the current character is equal to the target letter
            if (word.charAt(i) == targetLetter) {
                isPresent = true;
                break;
            }
        }

        // Print the result
        if (isPresent) {
            System.out.println("The letter '" + targetLetter + "' is present in the word.");
        } else {
            System.out.println("The letter '" + targetLetter + "' is not present in the word.");
        }
    }
}
```

In this program, we define the word "Umbrella" and the target letter 'e'. We initialize a boolean variable `isPresent` as `false` to track whether the target letter is found in the word.

We iterate through each character of the word using a for loop. Inside the loop, we check if the current character (`word.charAt(i)`) is equal to the target letter (`targetLetter`). If a match is found, we set `isPresent` to `true` and break out of the loop.

Finally, we print the result based on the value of `isPresent`.

The output of this program will be:

The letter 'e' is present in the word.

Since the letter 'e' is present in the word 'Umbrella', the program correctly identifies its presence.

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

ANSWER:-

In Java, the string constant pool is a special area of memory that is part of the Java Runtime Environment (JRE). It is a storage area where string literals and interned strings are stored. The string constant pool is located in the heap memory, which is the same memory area where objects are allocated.

The heap memory is a region of memory used for dynamic memory allocation in Java programs. It is shared among all threads and is used to store objects and their instance variables.

The string constant pool is created by the JVM when a Java program starts execution. It is a pool of unique string literals, meaning that if multiple strings with the same content are present in the program, they will be stored as a single copy in the string constant pool to save memory.

String literals, which are string values defined directly in the source code using double quotes ("..."), are automatically added to the string constant pool. Additionally, strings can be explicitly added to the string constant pool using the `String.intern()` method.

When a string is needed in the program, the JVM checks the string constant pool first. If a string with the same content is already present in the pool, a reference to that string is returned. If the string is not found in the pool, a new string object is created and added to the pool.

It's important to note that since the string constant pool is part of the heap memory, it is subject to garbage collection like other objects. Strings that are no longer referenced by any part of the program can be garbage collected and removed from the string constant pool.

Overall, the string constant pool is a specialized memory area within the heap memory where string literals and interned strings are stored to optimize memory usage and facilitate efficient string handling in Java.