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

**Answer**: In Java, the conditional operators are used to make decisions based on certain conditions and perform different actions accordingly. There are three conditional operators available in Java:

1. **Ternary Operator (?:):**

The ternary operator is the only conditional operator that takes three operands. It is also known as the conditional operator. It has the following syntax:

variable = (condition) ? expression1 : expression2;

If the condition is true, the value of `expression1` is assigned to the variable; otherwise, the value of `expression2` is assigned.

2. **Equality Operator (==):**

The equality operator is used to compare two values for equality. It checks whether the values of the two operands are equal or not. It has the following syntax:

(operand1 == operand2) If the values of `operand1` and `operand2` are equal, the expression evaluates to `true`; otherwise, it evaluates to `false`.

3**. Not-Equal Operator (!=):**

The not-equal operator is used to check whether two values are not equal. It checks if the values of the two operands are different or not. It has the following syntax:

(operand1 != operand2)

If the values of `operand1` and `operand2` are not equal, the expression evaluates to `true`; otherwise, it evaluates to `false`.

These conditional operators can be used in combination with other operators and control flow statements to create complex decision-making logic in Java programs.

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

**Answer:** Based on the number of operands, operators can be classified into three categories:

1. **Unary Operators:**

Unary operators are the operators that perform operations on a single operand. They include the following

- Unary Plus (+): Represents positive value.

- Unary Minus (-): Negates the value.

- Increment (++): Increases the value by 1.

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

- Logical Complement (!): Flips the boolean value.

2. **Binary Operators:**

Binary operators are the operators that perform operations on two operands. They include arithmetic, relational, logical, bitwise, and assignment operators. Some examples are:

- 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: Logical AND (&&), Logical OR (||).

- Bitwise Operators: Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^), Bitwise Complement (~).

- Assignment Operators: Assignment (=), Addition assignment (+=), Subtraction assignment (-=), Multiplication assignment (\*=), Division assignment (/=), etc.

3. **Ternary Operator:**

The ternary operator (?:) is the only operator that takes three operands. It is used for conditional expressions and involves evaluating a condition and choosing one of two expressions based on the result.

These different types of operators enable various operations and computations in Java programs.

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

**Answer:** The switch case statement in Java is used to perform different actions based on the value of a variable or an expression. It provides an efficient way to handle multiple possible values or conditions without using a series of if-else statements. The switch case statement has the following syntax:

switch (expression) {

case value1:

// Code to be executed if expression matches value1

break;

case value2:

// Code to be executed if expression matches value2

break;

...

case valueN:

// Code to be executed if expression matches valueN

break;

default:

// Code to be executed if expression doesn't match any case

break;

}

Here's how the switch case statement works:

1. The expression is evaluated, and its value is compared with each case value.

2. If a case value matches the expression value, the corresponding block of code is executed.

3. The `break` statement is used to exit the switch statement once a case is matched. This prevents the execution of subsequent cases.

4. If no case matches the expression value, the code block under the `default` label is executed (if present). The `default` case is optional.

5. After executing the code block for a matching case or the default case, the control is transferred outside the switch statement.

The switch case statement provides a more concise and readable way to handle multiple conditions compared to using a series of if-else statements. It is commonly used when there are a fixed number of possible values or conditions to be checked against a single expression.

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

**Answer:** Conditional statements in Java allow you to control the flow of your program based on certain conditions. They enable you to make decisions and execute different blocks of code depending on the evaluation of a Boolean expression.

Java provides three main types of conditional statements:

1. if statement:

The if statement is used to execute a block of code only if a specified condition is true. It has the following syntax:

if (condition) {

// Code to be executed if the condition is true

}

The condition is a Boolean expression that determines whether the code block should be executed. If the condition is true, the code block is executed. If the condition is false, the code block is skipped.

2. if-else statement:

The if-else statement allows you to specify two different blocks of code: one to be executed if the condition is true, and another to be executed if the condition is false. It has the following syntax:

if (condition) {

// Code to be executed if the condition is true

} else {

// Code to be executed if the condition is false

}

If the condition is true, the code block within the if statement is executed. If the condition is false, the code block within the else statement is executed.

3. switch statement:

The switch statement is used when you have multiple possible values or conditions to check against a single variable or expression. It allows you to execute different blocks of code based on the value of the expression. It has the following syntax:

switch (expression) {

case value1:

// Code to be executed if expression matches value1

break;

case value2:

// Code to be executed if expression matches value2

break;

default:

// Code to be executed if expression doesn't match any case

break;

}

The expression is evaluated, and its value is compared against the values specified in each case. If a match is found, the corresponding code block is executed. If no match is found, the code block within the default case is executed (if present).

The use of conditional statements in Java allows you to implement decision-making logic in your programs. You can perform different actions based on specific conditions, handle alternative scenarios, and control the program's flow to achieve desired behavior.

**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

}

The if-else statement consists of the following parts:

1. The keyword `if` followed by a set of parentheses `()`. Inside the parentheses, you provide the condition that will be evaluated. The condition should be a Boolean expression that determines whether the code block inside the if statement should be executed or not.

2. The code block inside the if statement is enclosed within curly braces `{}`. This block contains the code that will be executed if the condition evaluates to true.

3. The keyword `else` is followed by another code block enclosed within curly braces `{}`. This block contains the code that will be executed if the condition evaluates to false. The `else` block is optional.

Here's an example of the if-else statement in action:

int number = 10;

if (number > 0) {

System.out.println("The number is positive.");

} else {

System.out.println("The number is non-positive.");

}

In this example, if the `number` is greater than 0, the message "The number is positive" will be printed. Otherwise, the message "The number is non-positive" will be printed.

**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 these methods for string comparison:

1. `equals()` method:

The `equals()` method is used to compare the content of two strings and checks if they are equal. It returns a boolean value indicating whether the strings are equal or not. The syntax is as follows:

String str1 = "Hello";

String str2 = "World";

if (str1.equals(str2)) {

// Strings are equal

} else {

// Strings are not equal

}

In the example above, the `equals()` method is called on `str1` with `str2` as the argument. If the strings have the same content, the condition will evaluate to true, and the code block inside the `if` statement will be executed.

2. `compareTo()` method:

The `compareTo()` method is used to perform lexicographic comparison of two strings. It compares the strings based on their Unicode values and returns an integer value indicating their relative ordering. The syntax is as follows:

String str1 = "Apple";

String str2 = "Banana";

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 greater than str2

}

In the example above, the `compareTo()` method is called on `str1` with `str2` as the argument. The `compareTo()` method returns an integer value. If the value is 0, it means the strings are equal. If the value is less than 0, `str1` is lexicographically smaller than `str2`. If the value is greater than 0, `str1` is lexicographically greater than `str2`.

It's important to note that when comparing strings, you should use the `equals()` method for content comparison, rather than using the `==` operator, which checks for reference equality.

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

**Answer:** In Java, strings are immutable, which means their values cannot be changed once they are created. However, there is a mutable alternative to the immutable `String` class called `StringBuilder`.

`StringBuilder` is a class provided by Java that allows you to create and modify mutable strings. It provides methods for appending, inserting, deleting, and modifying characters in the string.

Here's an example that demonstrates the use of `StringBuilder`:

StringBuilder stringBuilder = new StringBuilder("Hello");

// Appending characters to the StringBuilder

stringBuilder.append(" World");

// Inserting characters at a specific position

stringBuilder.insert(5, " there");

// Deleting characters from the StringBuilder

stringBuilder.delete(5, 10);

// Modifying characters in the StringBuilder

stringBuilder.setCharAt(0, 'h');

// Converting StringBuilder to String

String result = stringBuilder.toString();

System.out.println(result); // Output: "helloWorld"

In the above example, we create a `StringBuilder` object and initialize it with the string "Hello". We then use various methods like `append()`, `insert()`, `delete()`, and `setCharAt()` to modify the string stored in the `StringBuilder` object.

Since `StringBuilder` is mutable, the modifications are performed on the same object, and the string can be changed without creating new objects. Finally, we convert the `StringBuilder` object to a regular `String` using the `toString()` method to obtain the final result.

The `StringBuilder` class is useful when you need to perform multiple modifications or concatenate strings efficiently in scenarios where the immutability of `String` objects would result in excessive object creation and memory usage.

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

**Answer:** class StringSorting{

public static void main(String[] args) throws Exception

{

// Custom string input

String str = "Gargi";

// Converting string into an array for computation

char arr[] = str.toCharArray();

// Nested loops for comparison of characters

// in above character array

char temp;

int i = 0;

while (i < arr.length) {

int j = i + 1;

while (j < arr.length) {

if (arr[j] < arr[i]) {

// Comparing the characters one by one

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

j += 1;

}

i += 1;

}

// By now loop is done means we have

// iterated the whole array

System.out.println(arr);

}

}

**Output:** Gagir

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

**'Umbrella'**

**Answer:** public class LetterCheck {

public static void main(String[] args) {

String word = "Umbrella";

char letter = 'e';

boolean isPresent = false;

// Loop through each character in the word

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

// Check if the current character is equal to the letter

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

isPresent = true;

break;

}

}

// Print the result

if (isPresent) {

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:** The letter 'e' is present in the word.

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

**Answer:** In Java, the string constant pool is a part of the runtime constant pool, which is a specific area of the Java Virtual Machine's (JVM) memory. More specifically, the string constant pool is located in the non-heap memory area known as the method area.

The method area is a shared memory region where the JVM stores class structures, method bytecode, constant pool, and other metadata required by the JVM at runtime. The string constant pool is a subsection within the method area and is used to store string literals, which are string objects created using the `String` class.

String literals are unique in Java, meaning that multiple references to the same string literal will point to the same memory location in the string constant pool. This is possible because strings are immutable in Java, allowing for optimization and memory efficiency.

It's important to note that the exact implementation details, including the location of the string constant pool, can vary across different JVM implementations. However, the general concept of the string constant pool being located in the method area remains consistent.