# **Assignment 2**

Q1. What are the Conditional Operators in Java?

There are three types of the conditional operator in Java:

1. Conditional AND

2. Conditional OR

3. Ternary Operator

### Conditional AND

The operator is applied between two Boolean expressions. It is denoted by the two AND operators (&&). It returns true if and only if both expressions are true, else returns false.

|  |  |  |
| --- | --- | --- |
| **Expression1** | **Expression2** | **Expression1 && Expression2** |
| True | False | False |
| False | True | False |
| False | False | False |
| True | True | True |

### Conditional OR

The operator is applied between two Boolean expressions. It is denoted by the two OR operator (||). It returns true if any of the expression is true, else returns false.

|  |  |  |
| --- | --- | --- |
| **Expression1** | **Expression2** | **Expression1 || Expression2** |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

### Ternary Operator

The meaning of **ternary** is composed of three parts. The **ternary operator (? :)** consists of three operands. It is used to evaluate Boolean expressions. The operator decides which value will be assigned to the variable. It is the only conditional operator that accepts three operands. It can be used instead of the if-else statement. It makes the code much easier, readable, and shorter.

**Syntax:**

variable = (condition) ? expression1 : expression2

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

**1. Arithmetic operators (+, -, \*, /, \*\*, %)**

Arithmetic operators perform mathematical operations such as addition and subtraction with operands. There are two types of mathematical operators: unary and binary. Unary operators perform an action with a single operand. Binary operators perform actions with two operands. In a complex expression, (two or more operands) the order of evaluation depends on precedence rules.

### **2. Unary Operators**

Unary operators need only one operand. They are used to increment, decrement, or negate a value.

* **–:** **Unary minus**, used for negating the values.
* **+:** **Unary plus** indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
* **++:** **Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.
* **Post-Increment:** Value is first used for computing the result and then incremented.
* **Pre-Increment:** Value is incremented first, and then the result is computed.
* **– –: Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.
* **Post-decrement:** Value is first used for computing the result and then decremented.
* **Pre-Decrement: The value** is decremented first, and then the result is computed.
* **!: Logical not operator**, used for inverting a boolean value.

### **3. Assignment Operator**

**‘=’** Assignment operator is used to assign a value to any variable. It has right-to-left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

variable **=** value;

### **4. Relational Operators**

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

variable **relation\_operator** value

### **5. Logical Operators**

These operators are used to perform “logical AND” and “logical OR” operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has “Logical NOT”, which returns true when the condition is false and vice-versa

Conditional operators are:

* **&&, Logical AND:** returns true when both conditions are true.
* **||, Logical OR:** returns true if at least one condition is true.
* **!, Logical NOT:** returns true when a condition is false and vice-versa

### **6. Ternary operator**

The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.

The general format is:

condition **?** if true **:** if false

### **7. Bitwise Operators**

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

* **&, Bitwise AND operator:** returns bit by bit AND of input values.
* **|, Bitwise OR operator:** returns bit by bit OR of input values.
* **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
* **~, Bitwise Complement Operator:** This is a unary operator which returns the one’s complement representation of the input value, i.e., with all bits inverted.

**8. Shift Operators**

* These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two. General format-
* number **shift\_op** number\_of\_places\_to\_shift;
* **<<, Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of two.
* **>>, Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect to dividing the number with some power of two.
* **>>>, Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

### **9. instanceof operator**

The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass, or an interface. General format-

object **instance of** class/subclass/interface

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

The Java *switch statement* executes one statement from multiple conditions. It is like [if-else-if](https://www.javatpoint.com/java-if-else) ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.

**Syntax:**

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**; //optional
5. **case** value2:
6. //code to be executed;
7. **break**; //optional
8. ......
10. **default**:
11. code to be executed **if** all cases are not matched;
12. }

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

Java has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false

## The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

### Syntax

if (*condition*) {  
 *// block of code to be executed if the condition is true*  
*}*

### Example

if (20 > 18) {  
 System.out.println("20 is greater than 18");  
}

## The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

### Syntax

if (*condition*) {  
 *// block of code to be executed if the condition is true*  
*}* else {  
 *// block of code to be executed if the condition is false*  
*}*

### Example

int time = 20;  
if (time < 18) {  
 System.out.println("Good day.");  
} else {  
 System.out.println("Good evening.");  
}  
// Outputs "Good evening."

## The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

### Syntax

if (*condition1*) {  
 *// block of code to be executed if condition1 is true*  
*}* else if (*condition2*) {  
 *// block of code to be executed if the condition1 is false and condition2 is true*  
*}* else {  
 *// block of code to be executed if the condition1 is false and condition2 is false*  
*}*

### Example

int time = 22;  
if (time < 10) {  
 System.out.println("Good morning.");  
} else if (time < 18) {  
 System.out.println("Good day.");  
} else {  
 System.out.println("Good evening.");  
}  
// Outputs "Good evening."

Q5. What is the syntax of if else statement?

## The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

### Syntax

if (*condition1*) {  
 *// block of code to be executed if condition1 is true*  
*}* else if (*condition2*) {  
 *// block of code to be executed if the condition1 is false and condition2 is true*  
*}* else {  
 *// block of code to be executed if the condition1 is false and condition2 is false*  
*}*

### Example

### **public** **class** IfElseExample {

**public** **static** **void** main(String[] args) {

//defining a variable

**int** number=13;

//Check if the number is divisible by 2 or not

**if**(number%2==0){

System.out.println("even number");

}**else**{

System.out.println("odd number");

}

}

}

Q6. How do you compare two strings in Java?

## **Method 1: String equals() method**

The **Java String equals() method** compares two string objects for equal string values.

### **Syntax:**

public boolean equals(Object anObject)

### **equals() method parameters:**

anObject – The argument string for comparing strings.

### **equals() method return types**

* Returns true if the string literal provided is the same as the first string.
* Returns false if the first string is not the same as the argument string.

### **Example:**

public class CompareTwoStrings {

public static void main(String[] args) {

String stringOne = "Guru";

String stringTwo = "Guru";

System.out.println("is Guru equal to Guru: "+stringOne.equals(stringTwo));

String stringThree = "GURU";

System.out.println("is Guru equal to GURU: "+stringOne.equals(stringThree));

}

}

## **Method 2: String compareTo() method**

The **Java String compareTo()** method compares two strings in alphabetic order. It is usually referred to as lexicographic order.

### **Syntax:**

public int compareTo(String str)

### **Example:**

public class CompareTwoStrings {

public static void main(String[] args) {

String stringOne = "Guru99";

String stringTwo = "Guru99";

System.out.println("is Guru99 equal to Guru99: " + stringOne.compareTo(stringTwo));

String stringThree = "GURU99";

System.out.println("is Guru99 equal to GURU99: " + stringOne.compareTo(stringThree));

}

}

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

**Mutable** means changing over time or that can be changed. In a **mutable string**, we can change the value of the string and **JVM** doesn’t create a new object. In a **mutable string**, we can change the value of the string in the same object.  
To create a **mutable string in java**, Java has two classes **StringBuffer** and **StringBuilder** where the **String class** is used for the **immutable string**.

To create a **mutable string**, we can use **StringBuffer** and **StringBuilder class**. Both classes create a **mutable object** of string but which one we should use totally depends on the scenario.

Suppose you want to work in a multithreading environment and the string should be thread-safe then you should use the **StringBuffer class.** On the other hand, if you don’t want a multithreading environment then you can use **StringBuilder** is not.  
But when you consider performance first then **StringBuilder** is better in terms of performance as compared to **StringBuffer**.

public class MutableString  
{  
 public static void main (String[] args)  
 {  
 StringBuffer str1 = new StringBuffer("JavaGoal");  
 StringBuilder str2 = new StringBuilder("Learning");  
   
 System.out.println("Value of str1 before change :" + str1);  
 System.out.println("Value of str2 before change :" + str2);  
   
 str1.append(".com");  
 str2.append(" website");  
   
 System.out.println("Value of str1 after change :" + str1);  
 System.out.println("Value of str2 after change :" + str2);  
 }  
}

Q8. Write a program to sort a String Alphabetically

class ExampleAlphaOrder {

public static void main(String[] args)

{

int n = 4;

String names[]= { "Rahul", "Ajay", "Gourav", "Riya" };

String temp;

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

if (names[i].compareTo(names[j]) > 0) {

temp = names[i];

names[i] = names[j];

names[j] = temp;

}

}

}

System.out.println("The names in alphabetical order are: ");

for (int i = 0; i < n; i++) {

System.out.println(names[i]);

}

}

}

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

'Umbrella'.

**class** Check\_Letter  
{  
 public **static** **void** main(String[] args)  
 {  
 String str = "umbrella";  
 boolean pre = **false**;  
 for(**int** i = 0;i<str.length();i++)  
 {  
 if(str.charAt(i) == 'e')  
 {  
 pre=**true**;  
 break;  
 }  
 }  
 System.out.println(pre);  
 }  
}

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

Memory?

The Java string constant pool is an area in heap memory where Java stores literal string values. The heap is an area of memory used for run-time operations. When a new variable is created and given a value, Java checks to see if that exact value exists in the pool or not.

### Memory Allocation in the String Pool

The String Pool is empty by default, and it is maintained privately by the String class.

When we create a string literal, the JVM first checks that literal in the String Constant Pool. If the literal is already present in the pool, its reference is stored in the variable.

However, if the string literal is not found, the JVM creates a new string object in the String Constant Pool and returns its reference.