Programming in Java

Java - Basic Operators

- The Relational Operators:

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The outcome of these operations is a **boolean** value. The relational operators are most frequently used in the expressions that control the if statement and the various loop statements.

Example:

```
public class Test {
      public static void main (String args[]) {
            int a = 10;
            int b = 20;
            System.out.println("a == b = " + (a == b) );
            System.out.println("a != b = " + (a != b) );
            System.out.println("a > b = " + (a > b) );
            System.out.println("a < b = " + (a < b) );
            System.out.println("b >= a = " + (b >= a) );
            System.out.println("b <= a = " + (b <= a) );
      }
}
```

- The Logical Operators:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	(x < 5) && (x < 10)
||	Logical or	Returns true if one of the statements is true	(x < 5) || (x < 4)
!	Logical not	Reverse the result, returns false if the result is true	!((x < 5) && (x < 10))

Example:

```
public class Test {
      public static void main(String args[]) {
            boolean a = true;
            boolean b = false;
            System.out.println("a && b = " + (a&&b));
            System.out.println("a || b = " + (a||b) );
            System.out.println("!(a && b) = " + !(a && b));
} }
```

Precedence of Java Operators

Precedence	Operator	Associativity
Postfix	( ) , [ ] , . (dot)	Left to right
unary	- (unary), + (unary) , ! , ~ , (type), ++, --	Right to left
Multiplicative	* , / , %	Left to right
Additive	+ , -	Left to right
Shift	>> , >>> , <<	Left to right
Relational	> , => , < , <=	Left to right
Equality	== , !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Assignment	= , += , -= , *= , /= , %= , >>= , <<= , &= , ^= , |=	Right to left
Comma	,	Left to right

Java Character Escape Sequence

In java, a character with a backslash(\) just before it is called an escape character or escape sequence.

A String represents a sequence of characters. These characters can be alphabets, numerals, punctuation marks, etc. While creating a String in Java the entire sequence of characters must be placed in quotation marks.

But what if the requirement is that this String must contain quotation marks i.e quotes within quotes. In this case compiler gets confused while interpreting the interior quotes of the String

To make things simple for the compiler we need to let it know when a quotation mark is a command (create a String) or when it is simply a character (display a word or a phrase along with quotation marks).

So, to solve the above problem we need **character escaping**. This is achieved using a special symbol \ **(backslash)** .

There are a total of 8 escape characters or escape sequences in java to pefrom different tasks.

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Example:

```
public class JavaEscape {
    public static void main(String args[]) {
        // Code for escapesequence \t
        System.out.println("Java is\t awesome");
        // Code for escapesequence \b
        System.out.println("Java is\b awesome");
        // Code for escapesequence \n
```

```
        System.out.println("Java is\n awesome");
        // Code for escapesequence \r
        System.out.println("Java is\r awesome");
        // Code for escapesequence \f
        System.out.println("Java is\f awesome");
        // Code for escapesequence \'
        System.out.println("Java \'is\' awesome");
        // Code for escapesequence \"
        System.out.println("Java \"is\" awesome");
        // Code for escapesequence \\
        System.out.println("\\Java is awesome");
    }
}
```

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements : if statements and switch statement.
2. Loop statements: do while loop, while loop, for loop, and for-each loop
3. Jump statements: break statement and continue statement

Decision Making statements:

The if statement

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. Syntax of if statement is given below.

```
if(condition) {
        statement 1; //executes when condition is true
}
```

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if (condition) {
        statement 1; //executes when condition is true
        }
else {
        statement 2; //executes when condition is false
        }
```

We can also use **else if** to add additional conditions to the if statement

Example:

```
public class Student {
      public static void main(String[] args) {
            String city = "Baghdad";
            if(city == "Basra") {
                  System.out.println("city is Basra ");
            }else if (city == "Erbil") {
                  System.out.println("city is Erbil ");
            }else if(city == " Diyala ") {
                  System.out.println("city is Diyala ");
            }else {
                  System.out.println(city);
            }
      }
}
```

The switch statement

In Java, switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable that is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the cases doesn't match the value of expression. It is optional.

- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

The syntax to use the switch statement is given below.

```
switch(expression) {
  case x:
   // code block
   break;
  case y:
   // code block
   break;
  default:
   // code block
}
```

Example:

```
public class WeekDays {
public static void main(String[] args) {
      int day = 4;
      switch (day) {
        case 1:
          System.out.println("Monday");
          break;
        case 2:
          System.out.println("Tuesday");
          break;
        case 3:
          System.out.println("Wednesday");
          break;
        case 4:
          System.out.println("Thursday");
          break;
        case 5:
          System.out.println("Friday");
          break;
        case 6:
          System.out.println("Saturday");
```

```
            break;
          case 7:
            System.out.println("Sunday");
            break;
      }
}
}
```

Loop Statements

Loop statements are used to execute a set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

- for loop

for loop in Java is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we know exactly the number of times, we want to execute the block of code.

```
for (initialization, condition, increment/decrement) {
//block of statements
}
```

Example:

```
public class Calculation {
      public static void main(String[] args) {
            int sum = 0;
            for(int j = 1; j<=10; j++) {
                  sum = sum + j;
            }
            System.out.println("The sum of first 10 natural numbers is "
            + sum);
} }
```

for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
for (data_type var : array_name/collection_name){
//statements
}
```

Example:

```
public class Test {
	public static void main(String[] args) {
		String[] names = {"Java","C","C++","Python","JavaScript"};
		System.out.println("Printing the content of the array
		names:\n");
		for(String name:names) {
			System.out.println(name);
		}
	}
}
```

- while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
while(condition){
//looping statements
}
```

Example:

```
public class Calculation {
	public static void main(String[] args) {
		int i = 0;
```

```
            System.out.println("Printing the list of first 10 even
            numbers \n");
            while(i<=10) {
                  System.out.println(i);
                  i = i + 2;
            }
      }
}
```

- do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iterations is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
//statements
} while (condition);
```

Example:

```
public class Calculation {
      public static void main(String[] args) {
            int i = 0;
            System.out.println("Printing the list of first 10 even
            numbers \n");
            do {
                  System.out.println(i);
                  i = i + 2;
            } while(i<=10);
      }
}
```

***Note**: any variable declared inside a loop is local to that code block. Meaning that it cannot be used outside that block without being declared again.

Example:

```
class Test {
```

```
    public static void main(String args[])
    {
        for (int x = 0; x < 4; x++)
        {
            System.out.println(x);
        }
         // printing x outside the loop
        System.out.println(x);
    }
}
```

This will result in an error.

Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

- **break statement**

The break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

- **continue statement**

Unlike break statement, the continue statement doesn't break the loop, whereas it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Example:

```
public class JumpStatements {
     public static void main(String[] args) {
        for (int i=0; i<=100; i++) {
          if (i%3==0)
               continue;
          if (i == 10)
               break;
       System.out.println(i);
       } }}
```