ASSIGNMENT 1

OPERATORS

1. **Arithmetic**
   Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Example:

```
class Main {
  public static void main(String[] args) {

    // declare variables
    int a = 12, b = 5;

    // addition operator
    System.out.println("a + b = " + (a + b));

    // subtraction operator
    System.out.println("a - b = " + (a - b));

    // multiplication operator
    System.out.println("a * b = " + (a * b));

    // division operator
    System.out.println("a / b = " + (a / b));

    // modulo operator
    System.out.println("a % b = " + (a % b));
  }
}
```

Output:

```
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
```

2. **Relational**

Relational operators are used to check the relationship between two operands.

Example 1:

```
// check if a is less than b
a < b;
```

Here, < operator is the relational operator. It checks if a is less than b or not.
It returns either true or false.

Operator	Description	Example
==	Is Equal To	3 == 5 returns false
!=	Not Equal To	3 != 5 returns true
>	Greater Than	3 > 5 returns false
<	Less Than	3 < 5 returns true
>=	Greater Than or Equal To	3 >= 5 returns false
<=	Less Than or Equal To	3 <= 5 returns true

Example 2:

```
class Main {
  public static void main(String[] args) {

    // create variables
    int a = 7, b = 11;
    // value of a and b
    System.out.println("a is " + a + " and b is " + b);

    // == operator
    System.out.println(a == b);  // false

    // != operator
    System.out.println(a != b);  // true

    // > operator
    System.out.println(a > b);  // false

    // < operator
    System.out.println(a < b);  // true

    // >= operator
    System.out.println(a >= b);  // false

    // <= operator
    System.out.println(a <= b);  // true
  }
}
```

3. **<u>Bitwise</u>**

   Bitwise operators in Java are used to perform operations on individual bits. For example,

   Bitwise complement Operation of 35:

```
35 = 00100011 (In Binary)
~ 00100011
________
  11011100  = 220 (In decimal)
```

   Here, ~ is a bitwise operator. It inverts the value of each bit (0 to 1 and 1 to 0).

   The various bitwise operators present in Java are:

Operator	Description
~	Bitwise Complement

<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

4. **<u>Logical</u>**

Logical operators are used to check whether an expression is true or false. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	true only if both expression1 and expression2 are true
|| (Logical OR)	expression1 || expression2	true if either expression1 or expression2 is true
! (Logical NOT)	!expression	true if expression is false and vice versa

Example:

```
class Main {
 public static void main(String[] args) {

   // && operator
   System.out.println((5 > 3) && (8 > 5));  // true
   System.out.println((5 > 3) && (8 < 5));  // false

   // || operator
   System.out.println((5 < 3) || (8 > 5));  // true
   System.out.println((5 > 3) || (8 < 5));  // true
   System.out.println((5 < 3) || (8 < 5));  // false
```

```
    // ! operator
    System.out.println(!(5 == 3));  // true
    System.out.println(!(5 > 3));  // false
  }
}
```

5. **Assignment**

   Assignment operators are used in Java to assign values to variables.
   For example,
   int age;
   age = 5;
   Here, = is the assignment operator. It assigns the value on its right to the variable on its left. That is, 5 is assigned to the variable age.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Example:

```
class Main {
  public static void main(String[] args) {

    // create variables
    int a = 4;
    int var;

    // assign value using =
    var = a;
    System.out.println("var using =: " + var);

    // assign value using =+
    var += a;
    System.out.println("var using +=: " + var);

    // assign value using =*
    var *= a;
    System.out.println("var using *=: " + var);
  }
}
```

Output:

```
var using =: 4
var using +=: 8
var using *=: 32
```

6. **Conditional**

   In Java, conditional operators check the condition and decides the desired result on the basis of both conditions.

   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 more easy, readable, and shorter.

   Syntax:

   variable = (condition) ? expression1 : expression2

   The above statement states that if the condition returns true, expression1 gets executed, else the expression2 gets executed and the final result stored in a variable.

   Example:

```
public class TernaryOperatorExample {
  public static void main(String args[]) {
  int x, y;
  x = 20;
  y = (x == 1) ? 61: 90;
 System.out.println("Value of y is: " +  y);
  y = (x == 20) ? 61: 90;
  System.out.println("Value of y is: " + y);
}
}
```

Output:

```
Value of y is: 90
Value of y is: 61
```

OPERATOR PRECEDENCE

Operator precedence determines the order in which the operators in an expression are evaluated.

Operators	Precedence
postfix increment and decrement	++ --
prefix increment and decrement, and unary	++ -- + - ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>=

CONTROL STATEMENTS

1. Selection

The selection statement means the statements are executed depending upon a condition.This statement is also called a decision making statement because it helps in making decisions about which set of statements are to be executed.

- **if**

  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. In Java, there are four types of if-statements given below.

- **Simple if statement**

  It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

  Syntax:

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

Example:

```
public class Student {
  public static void main(String[] args) {
      int x = 10;
      int y = 12;
      if(x+y > 20) {
            System.out.println("x + y is greater than 20");
      }
   }
 }
```

Output:

```
x + y is greater than 20
```

- **if-else statement**

  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
}
```

Example:

```
public class Student {

     public static void main(String[] args) {

            int x = 10;
             int y = 12;
            if(x+y < 10) {
                    System.out.println("x + y is less than      10");
             }
             else {
                  System.out.println("x + y is greater than 20");
              }
         }
     }
```

Output:

```
x + y is greater than 20
```

- **if-else-if ladder**

  The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

  Syntax:

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

```
 else if(condition 2) {
     statement 2; //executes when condition 2 is true
 }
else {
   statement 2; //executes when all the conditions are false
  }
```

Example:

```
public class Student {
  public static void main(String[] args) {
      String city = "Delhi";
        if(city == "Meerut") {
             System.out.println("city is meerut");
        }
      else if (city == "Noida") {
             System.out.println("city is noida");
        }
      else if(city == "Agra") {
             System.out.println("city is agra");
       }
       else {
            System.out.println(city);
      }
   }
}
```

Output:

```
Delhi
```

- **Nested if-statement**
  In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.

  Syntax:

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

Example:

```
public class Student {
public static void main(String[] args) {
String address = "Delhi, India";

if(address.endsWith("India")) {
if(address.contains("Meerut")) {
System.out.println("Your city is Meerut");
}else if(address.contains("Noida")) {
System.out.println("Your city is Noida");
}else {
System.out.println(address.split(",")[0]);
}
}else {
System.out.println("You are not living in India");
}
}
}
```

Output:

```
Delhi
```

- **switch**
  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 which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.
    - 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 case 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.

Syntax:

```
 switch (expression){
case value1:
statement1;
break;
.
.
.
case valueN:
 statementN;
 break;
default:
 default statement;
}
```

Example:

```
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Output:

```
2
```

2. **Iteration**

   In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

   - **do while**

     The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration 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.

     Syntax:

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

     Example:

     ```
     public class Calculation {
     public static void main(String[] args) {
     // TODO Auto-generated method stub
     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);
     }
     }
     ```

Output:

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

- **while**

  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.

  Syntax:

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

  Example:

  ```
  public class Calculation {
  public static void main(String[] args) {
  // TODO Auto-generated method stub
  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;
  ```

```
}
}
}
```

Output:

```
Printing the list of first 10 even numbers

0
2
4
6
8
10
```

- **for**
  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 exactly know the number of times, we want to execute the block of code.

  Syntax:

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

  Example:
  ```
  public class Calculattion {
  public static void main(String[] args) {
  // TODO Auto-generated method stub
  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);
  }
  }
  ```

Output:

```
The sum of first 10 natural numbers is 55
```

- **for each**

  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.

  Syntax:

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

  Example:

  ```
  public class Calculation {
  public static void main(String[] args) {
  // TODO Auto-generated method stub
  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);
  }
  }
  }
  ```

  Output:

  ```
  Printing the content of the array names:

  Java
  C
  C++
  Python
  JavaScript
  ```

3. **Jump**

   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**

     As the name suggests, 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.

     The break statement example with for loop:

     ```
     public class BreakExample {

     public static void main(String[] args) {
     // TODO Auto-generated method stub
     for(int i = 0; i<= 10; i++) {
     System.out.println(i);
     if(i==6) {
     break;
     }
     }
     }
     }
     ```

     Output:

     ```
     0
     1
     2
     3
     4
     5
     6
     ```

- **continue**
  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 ContinueExample {

public static void main(String[] args) {
// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {
continue;
}
System.out.println(j);
}
}
}

}
```

  Output:

```
0
1
2
3
5
1
2
3
5
2
3
5
```