Unit-2: Selections, Mathematical functions and loops

If statements, Two way, Nested if and multi-way if statements, Switch statements, Conditional Expressions, Common mathematical functions, While , do-while and for loop, nested loops, Keyword break and continue.

1. Explain if and two way if statements with illustrative example

If statements

- 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.
- 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 of if statement is given below.

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

Consider the following example in which we have used the **if** statement in the java code.

```
Student.java
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 15;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

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

Two way If statements (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
}
```

Consider the following example to check if the number is even or odd.

```
import java.util.Scanner;
public class EvenOdd {
public static void main(String[] args) {

      Scanner s = new Scanner(System.in);
      System.out.print("Enter a number: ");
      int num = s.nextInt();
      if(num % 2 == 0)
            System.out.println(num + " is even");
      else
            System.out.println(num + " is odd"); } }
```

Output:

```
Enter a number: 12
12 is even
```

2. Nested if

- In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.
- Syntax of Nested if-statement is given below.

```
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:

```
//Java nested if-else Program
import java.util.Scanner;
public class Main{
  public static void main(String []args){
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the age: ");
    int age=sc.nextInt();
    System.out.println("Enter the weight: ");
    int weight=sc.nextInt();
    if(age>=18){
      if(weight>50){
        System.out.println("The person is eligible to donate blood");
        }
       else{
        System.out.println("The person is not eligible to donate blood");
       }
     }
```

```
    else{
        System.out.println("Age must be greater than 18");
     }
   }
}
```

Output:

```
Enter the age: 24
Enter the weight: 49
The person is not eligible to donate blood
```

3. Multi-way if statements (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 of if-else-if statement is given below.

```
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
}
```

Consider the following example to print the greatest number among three numbers.

```
import java.util.Scanner;
public class Biggest_Number {
public static void main(String[] args){
int x, y, z;
Scanner s = new Scanner(System.in);
System.out.print("Enter the first number:");
x = s.nextInt();
System.out.print("Enter the second number:");
y = s.nextInt();
System.out.print("Enter the third number:");
z = s.nextInt();
if(x > y && x > z){
 System.out.println("Largest number is:"+x);
}
else if(y > z){
 System.out.println("Largest number is:"+y);
}
else{
```

```
        System.out.println("Largest number is:"+z);
      }
    }

  }
```

Output:

```
Enter the first number:10
Enter the second number:17
Enter the third number:15
Largest number is:17
```

4. What is the use of switch statement? Explain with a program.

- 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.
- The syntax to use the switch statement is given below.

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

Consider the following example to check if entered character is vowel or not using switch case

```
import java.util.*;
class SwitchVowel{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("\n Enter Character: ");
        char c = s.next().charAt(0);
        char z = Character.toUpperCase(c);
        switch(z) {
              case 'A':
              case 'E':
              case 'I':
              case 'O':
              case 'U': System.out.println(c+" is a Vowel.");
              break;
              default:
                    System.out.println(c+" is a Constant Character.");
              }
        }
 }
```

Output:

```
Enter Character: X
X is a Constant Character.
Enter Character: u
u is a Vowel.
Enter Character: c
c is a Constant Character.
```

5. Conditional Expressions

- In Java, conditional operators check the condition and decides the desired result on the basis of both conditions.
- There are three types of the conditional operator in Java
  - Conditional AND
  - Conditional OR
  - Ternary Operator

Operator	Symbol
Conditional or Logical AND	&&
Conditional or Logical OR	||
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

Let's create a Java program and use the conditional operator.

ConditionalOperatorExample.java

```
public class ConditionalOperatorExample{
public static void main(String args[]){
int x=5, y=4, z=7;
System.out.println(x>y && x>z || y<z);
System.out.println((x<z || y>z) && x<y);
}
}
```

Output:

```
true
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 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.

TernaryOperatorExample.java

```
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
```

6. Common mathematical functions

- The **java.lang.Math** class contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- This class provides mathematical functions in Java.

Example

```
public class JavaMathExample{
    public static void main(String[] args){
        double x = 28;
        double y = 4;
        // return the maximum of two numbers
        System.out.println("Maximum number of x and y is: " +Math.max(x, y));
        // return the square root of y
        System.out.println("Square root of y is: " + Math.sqrt(y));
        //returns 28 power of 4 i.e. 28*28*28*28
        System.out.println("Power of x and y is: " + Math.pow(x, y));
        // return the logarithm of given value
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));
        // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));
       }
}
```

Output:

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.60205999132796 24
```

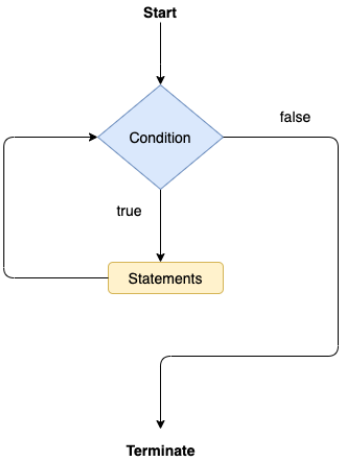
7. Explain the use of for loop, while loop and do-while loop using a program.

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
}
```

The flow chart for the while loop is given in the following image.



Consider the following example.

```
Calculation.java
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;
}
}
}
```

Output:

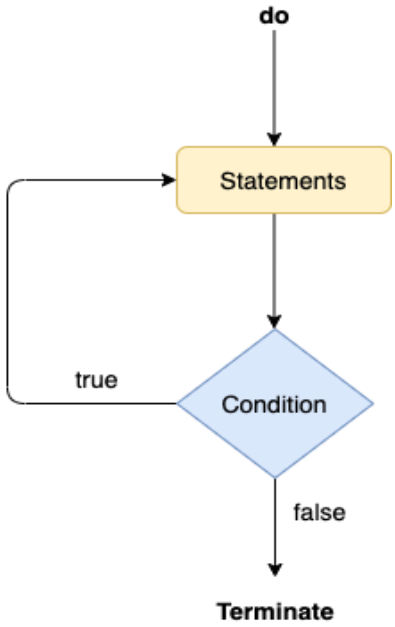
```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

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 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.
- The syntax of the do-while loop is given below.

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

flow chart of the do-while loop is given in the following image.



- Consider the following example to understand the functioning of the do-while loop in Java.

```
Calculation.java
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);
}
}
```

Output:

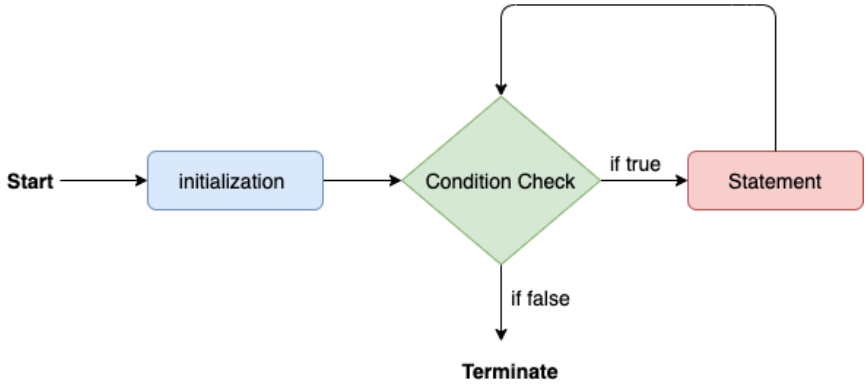
```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

For loop

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

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

- The flow chart for the for-loop is given below.



- Consider the following example to understand the proper functioning of the for loop in java.

```
Calculation.java
public class Calculattion {
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);
}
}
```

Output:

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

8. Nested loops

- If a loop exists inside the body of another loop, it's called a nested loop.
- Here, we are using a for loop inside another for loop.
- Consider the following example to display a multiplication table of 1 to 10 using nested loop.

```
public class Multiplication_Table{
      public static void main (String args[]){
            System.out.print("Multiplication table\n");
            for(int i=1; i<=10; i++){
                  for(int j=1; j<=10; j++){
                  System.out.print(i*j+"\t");
                  }
            System.out.println();
            }
      }
}
```

Output:

```
      Multiplication table
      1   2    3    4    5    6    7    8    9    10
      2   4    6    8    10   12   14   16   18   20
      3   6    9    12   15   18   21   24   27   30
      4   8    12   16   20   24   28   32   36   40
      5   10   15   20   25   30   35   40   45   50
      6   12   18   24   30   36   42   48   54   60
      7   14   21   28   35   42   49   56   63   70
      8   16   24   32   40   48   56   64   72   80
      9   18   27   36   45   54   63   72   81   90
      10  20   30   40   50   60   70   80   90   100
```

9. Explain break and continue statements with example.

Break Statement

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- The Java *break* statement is used to break loop or switch statement.
- It breaks the current flow of the program at specified condition.
- In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Example:

BreakExample.java

```
public class BreakExample {
public static void main(String[] args) {
    for(int i=1;i<=10;i++){
        if(i==5){
            //breaking the loop
            break;
        }
        System.out.println(i);
  }
 }
}
```

Output:

```
1
2
3
4
```

Continue Statement

- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately.
- It can be used with for loop or while loop.
- The Java *continue statement* is used to continue the loop.
- It continues the current flow of the program and skips the remaining code at the specified condition.
- In case of an inner loop, it continues the inner loop only.
- We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Example

ContinueExample.java

```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
//for loop
for(int i=1;i<=10;i++){
      if(i==5){
       //using continue statement
       continue;//it will skip the rest statement
       }
      System.out.println(i);
      }
    }
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

- As you can see in the above output, 5 is not printed on the console. It is because the loop is continued when it reaches to 5.