

Chapter 6. Statement and Control

The topics

| | |
|---|--|
| Java statements | |
| Control structure (flow of control) | |
| Decision making | |
| IF statement | |
| Different forms of if statement | |
| Switch statement | |
| Difference between switch and if-else | |
| Difference between if-else and ?:(conditional operator) | |
| Nested switch | |
| Iteration (looping) | |
| Components of looping statement | |
| Types of looping statements | |
| for loop | |
| while loop | |
| do while loop | |
| Different variations in looping | |
| Nested loops | |
| Jump statements | |
| Scope and visibility | |

1. Java statements

A Java statement is a single command executed by Java Interpreter. Java statements can be a single statement or a block of statements enclosed within a pair of balanced braces.

A single statement can have any one of the following types:

- | | |
|--|------------------------------------|
| i. Assignment statement – | int a=5; |
| ii. Arithmetic expression - | int a=0.5*b; |
| iii. Function/method call - | double p=Math.sqrt(9); |
| iv. Object creation using new operator - | Scanner sc=new Scanner(System.in); |
| v. Null or empty statement - | String s=null; |

A compound statement (a block of statements) can be of the following type:

- i. Selection/conditional statements
- ii. Iterative statements (Looping)
- iii. Jump statements
- iv. Exception handling statements (using try-catch block)

2. Control structure (flow of control)

When a program is executed, a control is passed to the first statement of that program and as the execution of the statements proceeds, the control flows down the program from one statement to another statement till the last statement reaches. And when the last statement's execution is over, the program ends there. This flow of control is of three types in a program. They are as follows –

- **Sequential flow**

In sequential flow, all the statements present in the program get executed in a sequence (in order of their appearance) in the program.

Example:

```
Statement1
Statement2
Statement3
```

Here, Statement 1 will be executed first, then Statement 2, and finally Statement 3.

- **Conditional flow (Decision-making statements)**

In conditional flow, there can be a set of statements that get executed and another statement leftover over unexecuted depending upon a condition.

Example:

```
Condition1
    Statement 1
Condition2
    Statement 2
Condition3
    Statement 3
```

Here out of the above 3 statements, only one statement will be executed depending upon which condition is satisfied.

- **Iterative flow (Looping statements)**

In looping or iterative flow, one or more set statements can be executed for a certain number of times. That means the statements will be repeated for a certain number of times depending upon a condition.

2.1 Decision making

Selection or decision-making statement allows selective execution of statements. It enables one to decide and take some of the several possible actions. Selection or decision-making statement means the execution of statements depending upon a test-condition. If the condition evaluates to true, then a course of action is followed otherwise, another course-of-action is followed.

There are three types of selection statements, they are :-

- a. **if statement**
- b. **switch statement**
- c. **conditional operator (ternary operator) statement**

2.1.1 IF statement

The if statement is a conditional branch statement. It is used to select one alternative statement out of two. The general form of the **if statement** is given below:

```
if (condition)
    statement 1;
else
    statement 2;
```

here, **if** and **else** are the keywords, the **statement** can be a single or a block of statements enclosed in curly braces and the **condition** is any test-expression that returns a true or false value.

2.1.2 Different forms of if statement

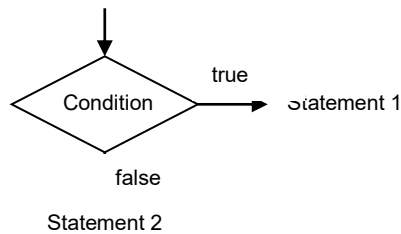
Following are the different forms of if statements that can be used in a Java program:

a. **if statement (single if statement)**

```
if (condition)
    statement 1;
```

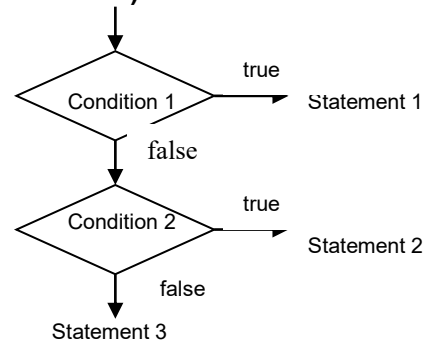
b. **if-else statement (most common form)**

```
if (condition)
    statement 1;
else
    statement 2;
```



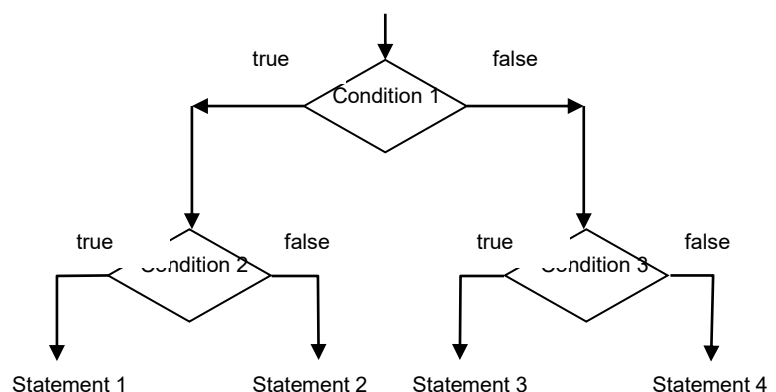
c. **if elseif else statement (else if ladder – multiple conditions)**

```
if (condition 1)
    statement 1;
else if (condition 2)
    statement 2;
else
    statement 3;
```



d. **nested if statement (if inside another if – multiple conditions)**

```
if (condition 1)
{
    if (condition 2)
        statement 1;
    else
        statement 2;
}
else
{
    if (condition 3)
        statement 3;
    else
        statement 4;
}
```



e. **multiple if statement:** here, we have multiple if statements and no else if or else.

```
if (condition 1)
    Statement 1;
if (condition 2)
    Statement 2;
if (condition 3)
    Statement 3;
```

Example programs on if structure:

Q1. WAP to print the absolute value of a number.

If N= -5, then the output will 5

If N=5, then there is no change

```
class Absolute
{
    public static void main (int N)
    {
        if(N<0)
            N= - N;
        System.out.println(N);
    }
}
```

Q2. WAP to print the largest of two unequal numbers

A=5, B=6, then output will 6

```
class Largest
{
    public static void main (int A, int B)
    {
        if(A>B)
            System.out.println(A);
        else
            System.out.println(B);
    }
}
```

Q3. WAP to accept two number and print they are equal or unequal.

import java.util.Scanner;

class EqualNumbers

```
{
    public static void main (String ar[])
    {
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter 2 nos.");
        int a=sc.nextInt( );
        int b=sc.nextInt( );
        if(a==b)
            System.out.println("Equal nos.");
        else
            System.out.println("Unequal nos.");
    }
}
```

Q4. WAP to accept two numbers and print which one is largest and which one is smallest.

(using if.. else if.. else structure)

```
import java.util.Scanner;
class LSNumbers
{
    public static void main (String ar[])
    {
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter 2 nos.");
        int a=sc.nextInt( );
        int b=sc.nextInt( );
        if(a>b)
            System.out.println(a+ "is largest &" + b+ "is smallest");
        else if(b>a)
            System.out.println(b+ "is largest &" + a+ "is smallest");
        else
            System.out.println("Both are equal nos.");
    }
}
```

Q5. WAP to accept two numbers and print which one is largest and which one is smallest.

(A) Nesting of if inside if block

```
import java.util.Scanner;
class LSNumbers
{
    public static void main (String ar[])
    {
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter 2 nos.");
        int a=sc.nextInt( );
        int b=sc.nextInt( );
        if(a!=b)//checking whether both the numbers are unequal or not
        {
            if(a>b)
                System.out.println(a+ "is largest &" + b+ "is smallest");
            else
                System.out.println(b+ "is largest &" + a+ "is smallest");
        }
        else
            System.out.println("Both are equal nos.");
    }
}
```

(B) Nesting of if inside else block

```
import java.util.Scanner;
class LSNumbers
{
    public static void main (String ar[])
    {
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter 2 nos.");
        int a=sc.nextInt( );
        int b=sc.nextInt( );
        if(a==b) //if both are equal then no further checking needed
            System.out.println("Both are equal nos.");
        else //otherwise checking continues
        {
            if(a>b)
                System.out.println(a+ "is largest &" + b+ "is smallest");
            else
                System.out.println(b+ "is largest &" + a+ "is smallest");
        }
    }
}
```

2.1.3 Switch statement

Switch statement is a multiple-branch selection statement. This selection statement successively tests the value of an expression against a list of integers or character constants. When a match is found, the statement associated with that constant is executed. If no match is found, then the default case is executed.

The general form of the switch statement is as follows:

```
switch (variable)
{
    case value1: statement 1;
                break;
    case value2: statement 2;
                break;
    case value3: statement 3;
                break;
    default:    statement 4;
}
```

here **switch**, **case**, **default**, and **break** are keywords, **value1**, **value2**, **value3** are the integer/character constants and **statement1**, **statement2**, and **statement3** are the statements to be executed. 'break' plays an important role in switch case as it transfers the control out of the switch block when a matched case being executed.

In the absence of a **break** statement, all the corresponding statements after the matched case will be executed in a sequence (including the default case statement). This process of execution of all the statements is called **Fall Through** in a switch case.

In switch, the comparison takes place for equality only. Switch case works better for menu-driven programming where there is a fixed set of options for selection and execution.

Some example programs on switch case statements

WAP to accept a digit and print it in words.

```
import java.util.Scanner;
class DigitPrint
{
    public static void main (String ar[])
    {
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter a digit");
        int a=sc.nextInt( );
        switch(a)
        {
            case 0: System.out.println("ZERO");
                    break;
            case 1: System.out.println("ONE");
                    break;
            case 2: System.out.println("TWO");
                    break;
        }
    }
}
```

```

        case 3: System.out.println("THREE");
        break;
        case 4: System.out.println("FOUR");
        break;
        case 5: System.out.println("FIVE");
        break;
        case 6: System.out.println("SIX");
        break;
        case 7: System.out.println("SEVEN");
        break;
        case 8: System.out.println("EIGHT");
        break;
        case 9: System.out.println("NINE");
        break;
        default: System.out.println("Invalid input");
    }
}
}

```

WAP to accept an alphabet and print it is vowel or not. (FALL THROUGH)

```

import java.util.Scanner;
class Vowel
{
    public static void main (String ar[])
    {
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter an alphabet");
        char a=sc.next().charAt(0);
        switch(a)
        {
            case 'a':
            case 'A':
            case 'e':
            case 'E':
            case 'i':
            case 'I':
            case 'o':
            case 'O':
            case 'u':
            case 'U':
                System.out.println("It is a vowel");
                break;
            default: System.out.println("Not a vowel");
        }
    }
}

```


In the above program fall through is taking place for every alphabet entered, if that is a vowel and the output will be the statement placed after the last case i.e. 'It is a vowel'. For rest of the cases, the default will get executed.

Similarity between Switch and If-elseif-else and to re-write a switch to if-elseif-else

- Both can work on comparison for equality
- Both are decision making statements
- Both are suitable for menu-driven programming

Program to convert a switch statement to if statement:

```
switch(c)
{
    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;
    default: System.out.println("Invalid day entered");
}
```

Re-writing the code in if-else structure:

```
if(c==1)
System.out.println("Monday");
else if(c==2)
System.out.println("Tuesday");
else if(c==3)
System.out.println("Wednesday");
else if(c==4)
System.out.println("Thursday");
else if(c==5)
System.out.println("Friday");
else if(c==6)
System.out.println("Saturday");
else if(c==7)
System.out.println("Sunday");
else
System.out.println("Invalid day entered");
```

2.1.4 Difference between switch and if-else

| If Else | Switch |
|---|--|
| 1. If else statement uses relational expression and logical expression as its test condition | 1. Switch statement tests for equality in the expression |
| 2. It works on all types of data types i.e. integer, floating point, character, String etc | 2. It works on only two data types – integer (int) and character (char) data type. |
| 3. The values of more than one variable or expression can be compared in the test expression. | 3. The value of the variable or expression can be compared against a set of possible values or constants |
| 4. It works on a range as any relational operation can be used as the test expression. | 4. It cannot work on a range as it checks for equality only. |

2.1.5 Difference between if-else and ?:(conditional operator)

| IF ELSE | ?:(conditional operator) |
|--|--|
| 1. If statement is flexible and can have more than one statement in a block. | 1. Conditional operator (?:) can give only one value at a given time. |
| 2. Nesting of if is easy to use because of the proper demarcation of each block. | 2. In its nested form the expression becomes more complex than if-else |
| 3. Any form of expression can be used inside if else block, even print statements. | 3. ?: is not so much flexible. Print statement cannot be used here. |

2.1.6 Nested switch

Like nested if, we can have nested switch statement also. The syntax will be like this:

```
switch (var1)
{
    case value1:
    switch (var2)
    {
        case value1: statement 1;
                    break;
        case value2: statement 2;
                    break;
        default    :      statement 3;
    }
    break;
    case value2:
    switch (var3)
    {
        case value1: statement 4;
                    break;
        case value2: statement 5;
                    break;
        default    :      statement 6;
    }
    break;
    default      :      statement 7;
}
}
```

2.2. Iteration (looping)

The iterative statement means the repetition of a set-of-statements for a certain number of times depending upon a condition. Till the condition is TRUE, the set-of-statements repeat again and again and when the condition becomes FALSE, the loop terminates.

What is a loop?

A loop is defined as a block of statements, which are repeatedly executed for a certain number of times depending upon certain conditions.

2.2.1 Components of looping statement

A looping statement consists of five components. They are as follows:

- i. Loop variable - is the variable/counter used in the loop to count the iteration.
- ii. Initialization - is the first step in which the starting value is assigned to the loop variable i.e, the starting value.
- iii. Condition - is the final value of the loop variable, where the loop comes to an end.
- iv. Re-initialization - is the numerical value added or subtracted to the loop variable in each (update) round of the loop and checked with the test-condition of the loop itself.
- v. Body - is the set of statements that is to be executed repeatedly.

2.2.2 Types of looping statements

There are three types of loop used in Java programming. They are –

- i. for loop
- ii. while loop
- iii. do while loop

2.2.3 for loop

The **for** loop statement comprises of three actions, placed in the **for** loop statement itself. The three actions are initialization, test-condition and re-initialization. The expressions are separated by semi-colon. The **for** loop allows to execute a set-of-statement until a certain condition is satisfied.

The general syntax of **for** loop –

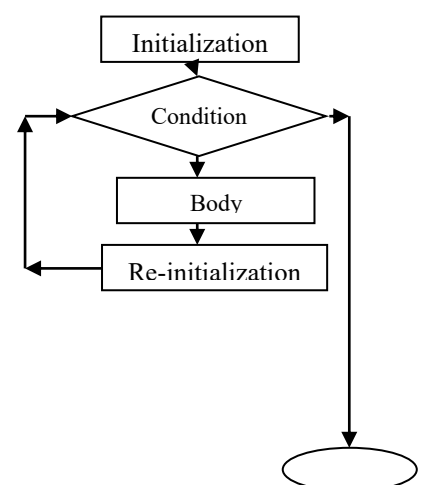
```
for(initialization; test-condition; re-initialization)
{
    set-of-statement; (body)
}
```

2.2.4 while loop

This is another kind of loop statement. It's general syntax is –

```
initialization;
while(test-condition)
{
    set-of-statements (body)
    re-initialization;
}
```

In this case, the test-condition may be any expression and the loop executes till the condition is true. When the condition becomes false, the loop terminates. This loop is also known as an **entry-controlled** loop because, if the condition is false at the beginning, the loop will not be executed for once.



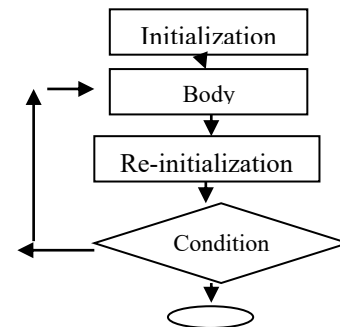
2.2.5 do while loop

This is another kind of loop statement. It's general syntax is –

```

initialization;
do
{
    set-of-statements (body)
    re-initialization;
} while(test-condition);

```



In this case, the test-condition is evaluated only at the time of exiting the loop i.e. even if the condition is false, the loop will be executed at least once. This loop is also known as an **exit-controlled** loop.

WAP to print 1st 10 natural nos.

Start value: 1
End value: 10
Step value: +1
Body: print the no.

using for loop:

```

for(int i=1; i<=10; i++)
{
    System.out.println(i);
}

```

WAP to print 1st ten even nos.

Start value: 0
End value: 20
Step value: +2
Body: print the even no.

using while loop:

```

int i=1;
while(i<=10)
{
    Sytem.out.println(i);
    i=i+1;
}

```

WAP to print 1st 10 odd no. in descending order

Start value: 19
End value: 1
Step value: -2
Body: print the odd no.

using do while loop:

```

int i=1;
do{
    System.out.println(i);
    i+=1;
}while(i<=10);

```

2.2.6 Different variations in looping

i. Multiple initializations & re-initialization in a for loop

In for loop, we can have multiple variables for initialization and/or re-initialization within the loop structure. Below is an example of the said variation -

```
for (int i=1, j=5; i*j !=0; i++, j--) //here we have two initialization and two re-initialization
{
    System.out.println( i*j );
}
```

ii. Finite loop & infinite loop

A finite loop is a loop where the number of iterations is fixed or finite i.e. the loop will end after a certain period of time. Every normal loop is a finite loop where the loop terminates when the test condition evaluates false.

Some examples of finite loops:

| <i>For loop</i> | <i>While loop</i> | <i>Do while loop</i> |
|--|--|---|
| <pre>for(int i=1;i<=5;i++) { statements }</pre> | <pre>int i=5; while(i>=1) { statements i--; }</pre> | <pre>int i=5; do { statements i--; }while(i>=1);</pre> |

An infinite loop is a loop where the number of iterations is infinite i.e. the loop will never end as because the test condition always evaluates true. It is also known as a never-ending loop. So if in a normal loop, we make the condition true always, the loop becomes an infinite loop.

Some examples of infinite loops:

| <i>For loop</i> | <i>While loop</i> | <i>Do while loop</i> |
|--|---|--|
| <pre>for(int i=1;i<=5;) { statements }</pre> | <pre>int i=5; while(i>=1) { statements }</pre> | <pre>int i=5; do { statements }while(i>=1);</pre> |
| <pre>for(; true ;) { statements }</pre> | <pre>while(true) { }</pre> | <pre>do { }while(true);</pre> |
| <pre>for(; ;) { statements }</pre> | | |

In all the above codes re-initialization statement is omitted from the loop, and thus test condition becomes true forever and thus the loops become infinite loops.

iii. **Empty loop & Non-empty loop**

An empty loop is a loop without any body i.e. this type of loop does not have any statement within its body to execute. Such loops are also called time-delay loops.

Some examples of empty loops:

| <i>For loop</i> | <i>While loop</i> | <i>Do while loop</i> |
|---|--|---|
| <pre>for(int i=1;i<=50;i++) { } </pre> | <pre>int i=50; while(i-->=1) { } </pre> | <pre>int i=5; do { } while(i++<=100); </pre> |
| <pre>for(int i=1;i<=50;i++); </pre> | <pre>while(i-->=1); </pre> | <pre>xxxxxxx </pre> |

In all the above loops, there are no codes in the body of the loop for execution. Thus they all are empty loops. Generally, empty loops are time-delay loops that are used to consume some time during the execution of the program without performing any specific task.

A non-empty loop is a normal loop with some statements as its body to execute.

Some examples of non-empty loops:

| <i>For loop</i> | <i>While loop</i> | <i>Do while loop</i> |
|---|--|--|
| <pre>for(int i=1;i<=5;i++) { System.out.println(i); } </pre> | <pre>int i=5,sum=0; while(i>=1) { sum+=i; i--; } </pre> | <pre>int i=1; do { System.out.println(i*5); i++; } while(i<=10); </pre> |

iv. **Known loop & Unknown loop**

A known loop is a loop where the number of iteration is known prior to enter the loop i.e. how many times the loop will run can be predicted before executing the loop.

Some examples of known loops:

| <i>For loop</i> | <i>While loop</i> | <i>Do while loop</i> |
|---|---|---|
| <pre>for(int i=1;i<=5;i++) { statements } </pre> | <pre>int i=5; while(i>=1) { statements i--; } </pre> | <pre>int i=5; do { statements i--; } while(i>=1); </pre> |

An unknown loop is a loop where the number of iterations is not known prior to entering the loop i.e. how many times the loop will run cannot be predicted before executing the loop.

Some examples of unknown loop:

| <i>For loop</i> | <i>While loop</i> | <i>Do while loop</i> |
|--|--|--|
| <pre>for(int i=1;i<=5;i++) { Statements if(i%2==0) i--; }</pre> | <pre>int i=5; while(i>=1) { Statements n=sc.nextInt(); if(n==5) i++; i--; }</pre> | <pre>int i=5; do { statements i--; } while(i>=1);</pre> |

v. **Entry-Controlled Loop & Exit –Controlled loop**

An **entry-controlled loop** is a loop where the test condition is evaluated before entering the loop or prior to loop execution. In this case, if the condition holds false at the beginning of the loop, the loop will not run at all. E.g of Entry controlled loops are – for loop and while loop.

An **exit controlled loop** is a loop where the test condition is evaluated at the end of the loop i.e. after the execution of the loop. In this case, if the condition holds false at the beginning of the loop, the loop will run at least once. E.g of Exit controlled loop – do while loop.

2.2.7 Nested Loops

A loop may contain another loop inside its body. Such types of loops are called nested loops. In a nested loop, the inner loop terminates before the outer loop and the inner loop repeats itself according to the iteration of the outer loop. In Java, nesting of loop can be done as many times as needed. Given below some examples of nested loops:

```
for(int i=1; i<=4; i++)//outer loop
{
    for(int j=1; j<=4; j++)//inner loop
    {
        System.out.print(j);//printing in the same line
    }
    System.out.println();//new line
}
```

In general, we use nested loops in pattern printing where outer loop is for the rows of the pattern and inner loop is the columns of the pattern.

Dry-run-chart:

| Outer loop | | Inner loop | | | | Outer loop | | Output |
|------------|-------|------------|-------|---------|-----|------------|-----|--------|
| I | I<=4 | J | J<=4 | PRINT J | J++ | PRINTLN | I++ | |
| 1 | True | 1 | True | 1 | 2 | | | 1234 |
| | | 2 | True | 2 | 3 | | | |
| | | 3 | True | 3 | 4 | | | |
| | | 4 | True | 4 | 5 | | | |
| | | 5 | False | X | X | newline | 2 | |
| 2 | True | 1 | True | 1 | 2 | | | 1234 |
| | | 2 | True | 2 | 3 | | | |
| | | 3 | True | 3 | 4 | | | |
| | | 4 | True | 4 | 5 | | | |
| | | 5 | False | X | X | newline | 3 | |
| 3 | True | 1 | True | 1 | 2 | | | 1234 |
| | | 2 | True | 2 | 3 | | | |
| | | 3 | True | 3 | 4 | | | |
| | | 4 | True | 4 | 5 | | | |
| | | 5 | False | X | X | newline | 4 | |
| 4 | True | 1 | True | 1 | 2 | | | 1234 |
| | | 2 | True | 2 | 3 | | | |
| | | 3 | True | 3 | 4 | | | |
| | | 4 | True | 4 | 5 | | | |
| | | 5 | False | X | X | newline | 5 | |
| 5 | False | X | X | X | X | X | X | |

1234
1234
1234
1234

Some examples of patterns using nested loop:

| | | |
|--|--|---|
| <pre> 1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7 int i, j, k; for(i=1; i<=4; i++) { k=i; for(j=1; j<=4; j++) { System.out.print(k+" "); k++; } System.out.println(); } </pre> | <pre> 1 3 5 7 2 4 6 8 3 5 7 9 4 6 8 10 int i, j, k; for(i=1; i<=4; i++) { k=i; for(j=1; j<=4; j++) { System.out.print(k+" "); k+=2; OR k=k+2; } } </pre> | <p>Try out:</p> <pre> 4 3 2 1 5 4 3 2 6 5 4 3 7 6 5 4 4 3 2 1 6 5 4 3 8 7 6 5 7 5 3 1 8 6 4 2 9 7 5 3 9 7 5 3 1 8 6 4 2 7 5 3 6 4 5 </pre> |
| <pre> 1 2 3 4 3 4 5 6 5 6 7 8 int i, j, k; for(i=1; i<=5; i=i+2) { k=i; for(j=1; j<=4; j++) { } } </pre> | <pre> 1 3 5 7 9 2 4 6 8 3 5 7 4 6 5 int i, j, k, p=9; for(i=1; i<=5; i++) { k=i; for(j=i; j<=p; j+=2) { System.out.print(k+" "); k+=2; } p--; System.out.println(); } </pre> | |

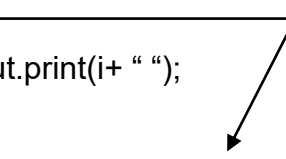
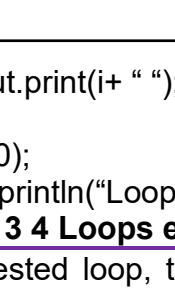
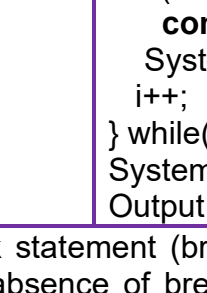
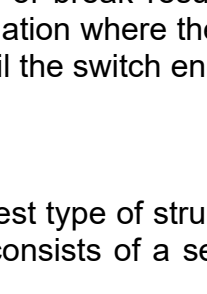
2.3 Jump statements

There are two jump statements used in Java programs on loops. They are – break and continue.

- **break** – this statement can appear in a loop as well as in a switch statement. In which ever statement it appears, it terminates that block statement and places the control to the next line outside that block statement.
- **continue** – this statement appears only in loops. continue statement abandons the current iteration of the loop by skipping over the rest of the statements in the loop body. It immediately transfer the control to the beginning of the loop once again.

Given below an example to show the working of break and continue statement:

| Use of break in a for loop | Use of continue in a for loop |
|---|--|
| <pre> for(int i=1; i<=10; i++) { if(i%5==0) break; System.out.print(i+ " "); } System.out.println("Loop ends"); Output : 1 2 3 4 Loops ends </pre> | <pre> for(int i=1; i<=10; i++) { if(i%5==0) continue; System.out.print(i+ " "); } System.out.println("Loop ends"); Output : 1 2 3 4 6 7 8 9 Loop ends (5 & 10 skipped) </pre> |

| <i>Use of break in a while loop</i> | <i>Use of continue in a while loop</i> |
|--|--|
| <pre>int i=1; while(i<=10) { if(i%5==0) break; System.out.print(i+ " "); i++; } System.out.println("Loop ends"); Output : 1 2 3 4 Loops ends</pre>  | <pre>int i=1; while(i<=10) { if(i%5==0) continue; System.out.print(i+ " "); i++; } System.out.println("Loop ends"); Output : 1 2 3 4 (Infinite loop) [here i++ is unreachable as it is after the continue statement in while loop]</pre>  |
| <i>Use of break in a do while loop</i> | <i>Use of continue in a do while loop</i> |
| <pre>int i=1; do { if(i%5==0) break; System.out.print(i+ " "); i++; } while(i<=10); System.out.println("Loop ends"); Output : 1 2 3 4 Loops ends</pre>  | <pre>int i=1; do { if(i%5==0) continue; System.out.print(i+ " "); i++; } while(i<=10); System.out.println("Loop ends"); Output : 1 2 3 4 (Infinite loop)</pre>  |

In case of nested loop, the break statement (break or continue) acts upon the loop where it is mentioned. In switch statement, absence of break results in **fall through**. In absence of break statement with a particular case, it is a situation where the control statement flows to the next case and so on until a break is encounter or until the switch ends. This is known as fall-through.

3. Scope and visibility

In Java programming, a block is the simplest type of structured statements that group a sequence of statements into a single statement. It consists of a sequence of statements within a balanced pair of braces.

Short answer questions (Output finding):

- The following public function is part of some class. Assume n is always positive. Write down the output of the following code of segment if n = 11. You have to show the working of the function calls.

```
void function(int n)
{
    if( n% 2 == 0)
        return 1;
    else
        return function( n/2 )* 10 + n%2;
}
```

- State the output of the following code segment. Also count and state how many times the loop will run and what will be the final value of k?

```
void main()
{
    int i = 1, j = 0, k = 0;
    while( i <= 1)
    {
        for( j=0; j<10; j++);
        do
        {
            System.out.println(k++);
        } while(k < 0);
        i++;
    }
}
```

- State the final value of q of the following program segment. Also show the dry run sequence:

```
int m, n, p, q=0;
for(m = 2; m<=3; ++m )
    for(n = 1; n<=m; ++n )
    {
        p = m + n - 1;
        if( p%3 == 0 )
            q += p;
        else
            q += p + 4;
    }
```

- State the output of the following program code:

```
class Num
{
    int k=0,sum;
    static int i=0;
    public static void display( int i )
    {
        if( i==0)
            sum = 0;
        for( k=1; k<i ; k=k*2)
            sum+=k;
        System.out.println(sum);
    }
}
```

5. Answer the following questions:-

- a. Define conditional statement. Write down the various types of conditional statements used in Java programming with complete syntax
- b. What is the significance of test-condition in an if statement? Explain with an example.
- c. Write down the similarity and difference between if else statement and ternary operator.
- d. What is the significance of break statement in a switch statement?
- e. What is fall through?
- f. What is dangling else? How can we resolve this problem?
- g. What is the significance of default clause in a switch statement?
- h. Write down the advantages and disadvantages of switch statement over if else if statement.
- i. What are iteration statements? Name the iteration statements provided by Java.
- j. What are the components of loop?
- k. State the difference between for loop, while loop and do while loop.
- l. What is an infinite loop? Give some forms of infinite loops.
- m. What is an empty loop? Why it is used in Java programming?
- n. Differentiate between break and continue statements inside a loop body with example.

6. Solve the following programs

- 1) Write a program to calculate commission for the salesmen. The commission is calculated according to following rates:

| Sales | Commission Rate | Sales continued for | Bonus |
|---------------|-----------------|---------------------|-------|
| 30001 onwards | 15% | 24 months or above | 30% |
| 22001-30000 | 10% | 18 – 23 months | 25% |
| 12001-22000 | 7% | 12 – 17 months | 20% |
| 5001-12000 | 3% | 5 – 11 months | 15% |
| 0-5000 | 0% | 2 – 4 months | 5% |

The program accepts the sales made by salesman and display the calculated commission and bonus.

- 2) Write a program to calculate and print the roots of a quadratic equation $ax^2+bx+c=0$. The coefficients of quadratic equation a, b, c are received as parameters.
- 3) Write a program to check whether the given number is Prime or not. Also print the next prime number of that number irrespective of that the given number is prime or not.
- 4) Write a program to find out all the 3-digit Armstrong numbers, along with their factors.
- 5) Write a program to calculate the following series:

$$x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \dots \pm \frac{x^n}{n!}$$

- 6) Write a program to print the following pattern, for n, where n will be given by the user: For n=5, the pattern will be -

```

*
*1*1*
*2*2*2*2*
*3*3*3*3*3*3*
*4*4*4*4*4*4*4*

```