FUNDAMENTAL PROGRAMMING STRUCTURES IN JAVA

Java Comments

The java comments are **statements that are not executed by the compiler** and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

Types of Java Comments

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

1) Java Single Line Comment

The single line comment is used to comment only one line. A single-line comment begins with a // and ends at the end of the line.

Syntax	Example
//Comment	//This is single line comment

2) Java Multi Line Comment

This type of comment must begin with /* and end with */. Anything between these two comment symbols is ignored by the compiler. A multiline comment may be several lines long.

Syntax	Example
/*Comment starts continues continues . . . Commnent ends*/	/* This is a multi line comment */

3) Java Documentation Comment

This type of comment is used to produce an HTML file that documents our program. The documentation comment begins with a /** and ends with a */.

Syntax	Example
/**Comment start * *tags are used in order to specify a parameter *or method or heading *HTML tags can also be used *such as <h1> * *comment ends*/	/** This is documentation comment */

DATA TYPES

Java is a **statically typed and also a strongly typed language. I**n Java, each type of data (such as integer, character, hexadecimal, etc.) is predefined as part of the programming language and all constants or variables defined within a given program must be described with one of the data types.

Data types represent the different values to be stored in the variable. In java, there are two categories of data types:

- Primitive data types
- Non-primitive data types

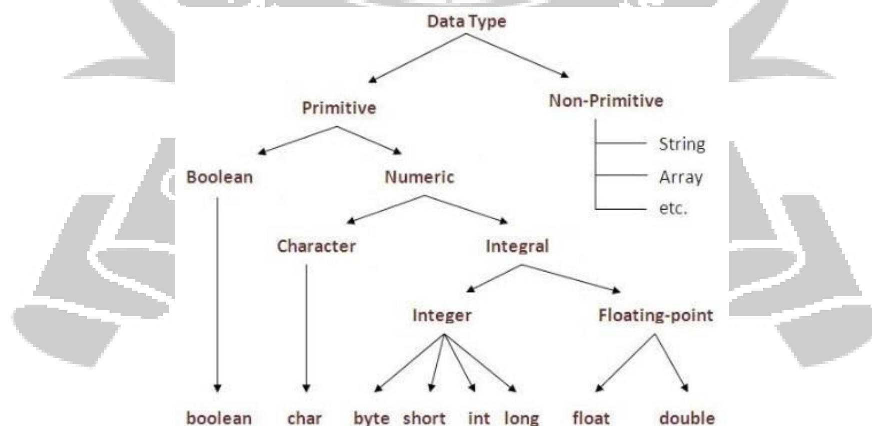


Figure: Data types in java

The Primitive Types

Java defines eight primitive types of data: **byte, short, int, long, char, float, double, and boolean.** The primitive types are also commonly referred to as simple types and they are grouped into the following four groups:

i) ***Integers*** - This group includes byte, short, int, and long. All of these are signed, positive and negative values. The width and ranges of these integer types vary widely, as shown in the following table:

Name	Width in bits	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Table: Integer Data Types

ii) ***Floating-point numbers*** – They are also known as real numbers. This group includes float and double, which represent single- and double-precision numbers, respectively. The width and ranges of them are shown in the following table:

Table: Floating-point Data Types

Name	Width in bits	Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

iii) ***Characters*** - This group includes char, which represents symbols in a character set, like letters and numbers. char is a 16-bit type. The range of a char is 0 to 65,536. There are no negative chars.

iv) ***Boolean*** - This group includes boolean. It can have only one of two possible values, true or false.

VARIABLES

A variable is the **holder that can hold the value while the java program is executed**. A variable is assigned with a datatype. It is name of ***reserved area allocated in memory***. In other words, it is a ***name of memory location***. There are three types of variables in java: local,instance and static.

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations thatcan be applied to the variable.

Before using any variable, it must be declared. The following statement expresses the basic form of a variable declaration –

datatype variable [= value][, variable [= value] ...] ;

Here data type is one of Java's data types and variable is the name of the variable. To declare more than one variable of the specified type, use a comma-separated list.

Example

```
int a, b, c;          // Declaration of variables a, b, and c.
int a = 20, b = 30; // initialization
byte B = 22;          // Declaratrion initializes a byte type variable B.
```

Types of Variable

There are three types of variables in java:

- local variable
- instance variable
- static variable

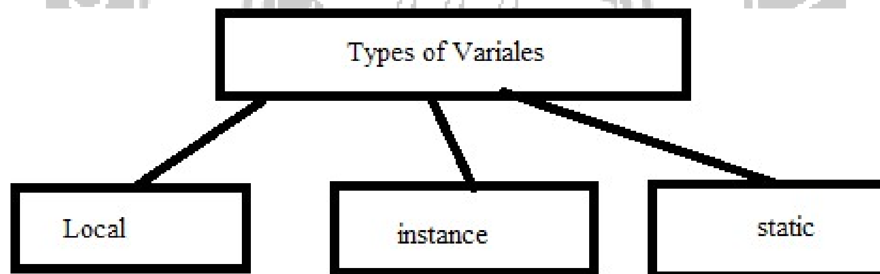


Fig. Types of variables

Local Variable

- Local variables are declared **inside the methods**, constructors, or blocks.
- Local variables **are created when the method, constructor or block is entered**
- Local variable **will be destroyed once it exits the method**, constructor, or block.
- Local variables **are visible only within the declared method**, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.
- Access specifier cannot be used for local variables.

Instance Variable

- A variable declared inside the class but outside the method, is called **instance variable**.Instance variables are declared in a class, but outside a method, constructor or any block.
- A slot for each **instance variable value is created when a space is allocated** for an object in the heap.
- Instance variables are created when an object is created with the use of the keyword **'new'** and destroyed when the object is destroyed.
- **Instance variables hold values that must be referenced by more than one method, constructor or block,** or essential parts of an object's state that must be present throughout the class.
- **Instance variables can be declared in class level before or after use.**
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. It is recommended to make these variables as private. However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values.
  - numbers, the default value is 0,
  - Booleans it is false,
  - Object references it is null.
- Values can be assigned during the declaration or within the constructor.
- **Instance variables cannot be declared as static**.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name.

ObjectReference.VariableName.

Static variable

- Class variables also known as static variables are declared with the **static keyword** ina class, but outside a method, constructor or a block.
- Only one copy of each class variable per class is created, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. **Constants are variables that are declared as public/private, final, and static.** Constant variables never change from their initial value.

- Static variables are stored in the **static memory**. It is rare to use static variables otherthan declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is same as instance variables. However, most **static variables are declared public** since they must be available for users of the class.
- Default values are same as instance variables.
  - numbers, the **default value is 0;**
  - Booleans, it is false;
  - Object references, it is null.
- Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables cannot be **local.**
- Static variables can be accessed by calling with the class name ClassName. VariableName.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

OPERATORS

Operator in java is a symbol that is used to perform operations. Java provides a rich set of operators to manipulate variables. For example: +, -, *, / etc.

All the Java operators can be divided into the following groups −

- ***Arithmetic Operators*** :

  Multiplicative : `* / %`

  Additive : `+ -`

- ***Relational Operators***

  Comparison : `< > <= >= instanceof`

  Equality : `== !=`

- ***Bitwise Operators***

  bitwise AND : `&`

  bitwise exclusive OR : `^`

  bitwise inclusive OR : `|`

  Shift operator: << >> >>>

- ***Logical Operators***

  logical AND : &&

  logical OR : ||

  logical NOT : ~ !

- ***Assignment Operators***: =
- ***Ternary operator***: ? :
- ***Unary operator***

  Postfix : *expr++ expr–*

  Prefix : ++*expr* −−*expr* +*expr* −*expr*

The Arithmetic Operators

Arithmetic operators are used **to perform arithmetic operations** in the same way as theyare used in algebra. The following table lists the arithmetic operators −

Example:

int A=10,B=20;

Operator	Description	Example	Output
+ (Addition)	Adds values A & B.	A + B	30
- (Subtraction)	Subtracts B from A	A - B	-10
* (Multiplication)	Multiplies values A & B	A * B	200
/ (Division)	Divides B by A	B / A	2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A	0

// Java program to illustrate arithmetic operators

```
public class Aoperators
{
  public static void main(String[] args)
  {
    int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
    String x = “Thank”, y = “You”;
    System.out.println(“a + b = “+(a +
    b)); System.out.println(“a - b = “+(a -
    b));
```

System.out.println("x + y = "+x + y);

*System.out.println("a * b = "+(a **

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

b));

System.out.println("a % b = "+(a % b));

}

}

The Relational Operators

The following relational operators are supported by Java language.

Example:

int A=10,B=20;

Operator	Description	Example	Output
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B)	true
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B)	true
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B)	true
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B)	true
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B)	true
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B)	true
instance of Operator	checks whether the object is of a particular type (class type or interface type) (Object reference variable) instanceof (class/interface type)	boolean result = name instanceof String;	True

```
// Java program to illustrate relational operators
public class operators
{
    public static void main(String[] args)
    {
        int a = 20, b = 10;
        boolean condition =
        true;
        //various conditional operators
        System.out.println("a == b :" + (a == ));
        System.out.println("a < b :" + (a < b));
        System.out.println("a <= b :" + (a <= ));
        System.out.println("a > b :" + (a > b));
        System.out.println("a >= b :" + (a >= ));
        System.out.println("a != b :" + (a != ));
        System.out.println("condition==true :" + (condition == true));
    }
}
```

Bitwise Operators

Java supports several bitwise operators, that can be applied to the integer types, long, int, short, char, and byte. Bitwise operator works on bits and performs bit-by-bit operation.

Example:

int a = 60,b = 13;

binary format of a & b will be as follows −

a = 0011 1100

b = 0000 1101

Bitwise operators follow the truth table:

a	b	a&b	a|b	a^b	~a
0	0	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	0
1	1	1	1	1	0

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

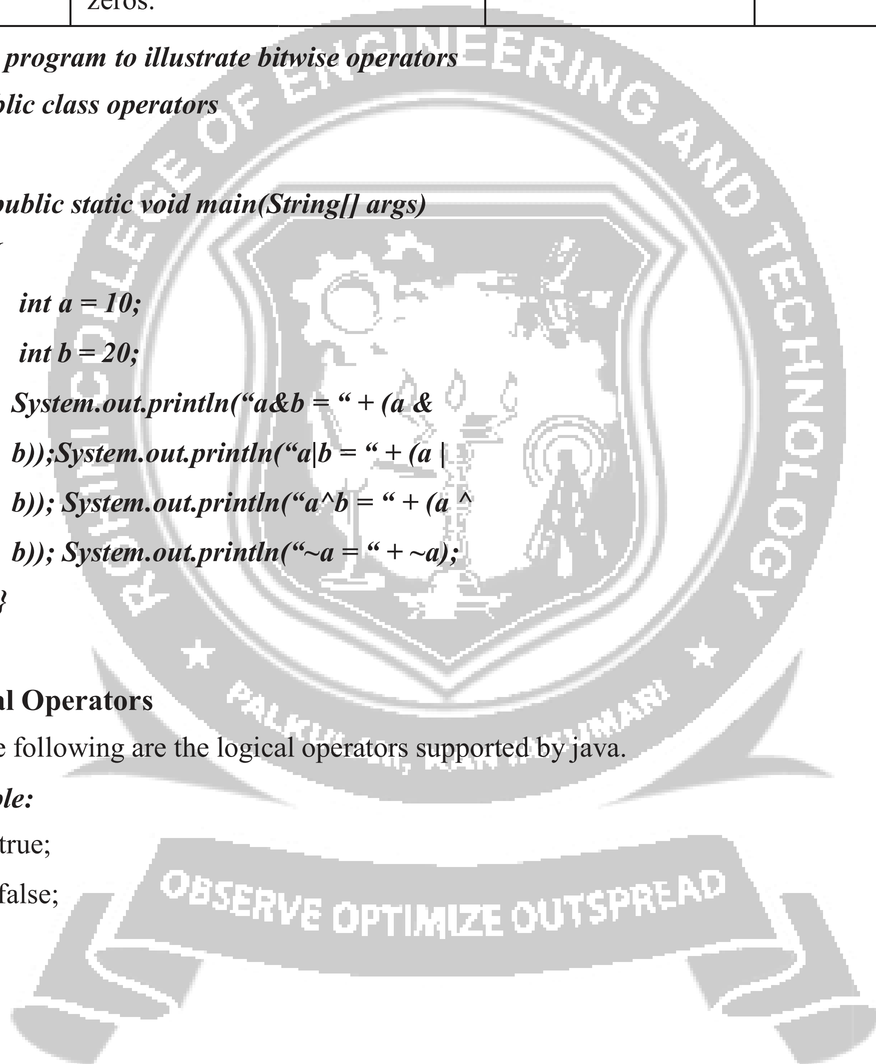
The following table lists the bitwise operators −

int A=60,B=13;

Operator	Description	Example	Output
& (bit-wise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is	12 (in binary form: 0000 1100)
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B)	61 (in binary form: 0011 1101)
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001	49 (in binary form: 0011 0001)
~ (bitwise compli-ment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.	-61 (in binary form: 1100 0011)
<< (left shift)	The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000	240 (in binary form: 1111 0000)

>> (right shift)	The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111	15 (in binary form: 1111)
>>> (zero fill right shift)	The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111	15 (in binary form: 0000 1111)

```
// Java program to illustrate bitwise operators
  public class operators
  {
    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 = " + ~a);
    }
  }
```



Logical Operators

The following are the logical operators supported by java.

Example:

A=true;

B=false;

Operator	Description	Example	Ouptput
&& (logical and)	If both the operands are non-zero, then the condition becomes true.	(A && B)	false
|| (logical or)	If any of the two operands are non-zero, then the condition becomes true.	(A || B)	true
! (logical not)	Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B)	true

Assignment Operators

The following are the assignment operators supported by Java.

Operator	Description	Example
= (Simple assignment operator)	Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+= (Add AND assignment operator)	It adds right operand to the left operand and assigns the result to left operand.	C += A is equivalent to C = C + A
-= (Subtract AND assignment operator)	It subtracts right operand from the left operand and assigns the result to left operand.	C -= A is equivalent to C = C – A
*= (Multiply AND assignment operator)	It multiplies right operand with the left operand and assigns the result to left operand.	C *= A is equivalent to C = C * A

/= (Divide AND assignment operator)	It divides left operand with the right operand and assigns the result to left operand.	C /= A is equivalent to C = C / A
%= (Modulus AND assignment operator)	It takes modulus using two operands and assigns the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

```
// Java program to illustrate assignment operators
public class operators
{
  public static void main(String[] args)
  {
    int a = 20, b = 10, c, d, e = 10, f = 4, g =
     9;c = b;
    System.out.println("Value of c = " +
    c);a += 1;
```

```
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a, b, e, f = " +a + "," + b + "," + e + "," + f);
    }
}
```

Ternary Operator

Conditional Operator (? :)

Since the conditional operator has three operands, it is referred as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as –

variable x = (expression) ? value if true : value if false

Following is an example −

Example:

```
public class example
{
public static void main(String args[])
{
int a,
b;a =
10;
b = (a == 0) ? 20: 30;
System.out.println( "b : " + b );
}
}
```

Unary Operators

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

Operator	Description
- Unary minus	negating the values
+ Unary plus	converting a negative value to positive
++ :Increment operator	incrementing the value by 1
— : Decrement operator	decrementing the value by 1
! : Logical not operator	inverting a boolean value

```
// Java program to illustrate unary operators
public class operators
{
public static void main(String[] args)
{
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
boolean condition =true;

c = ++a;
System.out.println("Value of c (++a) = " +c);
c = b++;
System.out.println("Value of c (b++) = " + );
c = --d;
System.out.println("Value of c (--d) = "+c);
c = --e;
System.out.println("Value of c (--e) = " + c);
System.out.println("Value of !condition =" + !condition);
}
}
```

Precedence of Java Operators

Operator precedence determines the grouping of operands in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator

For example, the following expression,

x = 10 + 5 * 2;

is evaluated. So, the output is 20, not 30. Because operator * has higher precedence than +.

The following table shows the operators with the highest precedence at the top of the table and those with the lowest at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	>() [] . (dot operator)	Left to right
Unary	>++ - - ! ~	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
Conditional	?:	Right to left
Assignment	>= += -= *= /= %= >>= <<= &= ^= |=	Right to left

CONTROL FLOW

Java Control statements control the flow of execution in a java program, based on data values and conditional logic used. There are three main categories of control flow statements;

Selection statements: if, if-else and switch.

Loop statements: while, do-while and for.

Transfer statements: break, continue, return, try-catch-finally and assert.

Selection statements

The selection statements checks the condition only once for the program execution.

If Statement:

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program.

The simple if statement has the following syntax:

if (<conditional expression>)

<statement action>

The following program explains the if statement.

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

The If-else Statement

The if/else statement is an extension of the if statement. If the condition in the if statement fails, the statements in the else block are executed. The if-else statement has the following syntax:

if (<conditional expression>)

<statement action>

else

<statement action>

The following program explains the if-else statement.

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

```
int a = 10, b =
20;if (a > b)
{
System.out.println("a > b");
}
else
{
System.out.println("b < a");
}
}
}
```

Switch Case Statement

The switch case statement is also called as multi-way branching statement with several choices. A switch statement is easier to implement than a series of if/else statements. The switch statement begins with a keyword, followed by an expression that equates to a no long integral value.

After the controlling expression, there is a code block that contains zero or more labeled cases. Each label must equate to an integer constant and each must be unique. When the switch statement executes, it compares the value of the controlling expression to the values of each case label.

The program will select the value of the case label that equals the value of the controlling expression and branch down that path to the end of the code block. If none of the case label values match, then none of the codes within the switch statement code block will be executed.

Java includes a default label to use in cases where there are no matches. A nested switch within a case block of an outer switch is also allowed. When executing a switch statement, the flow of the program falls through to the next case. So, after every case, you must insert a break statement.

The syntax of switch case is given as follows:

```
switch (<non-long integral expression>) {
        case label1: <statement1>
        case label2: <statement2>
        ...
        case labeln: <statementn>
        default: <statement>
        } // end switch
```

The following program explains the switch statement.

```
public class ProgramSwitch
{
public static void main(String[] args)
{
int a = 10, b = 20, c =
30;int status = -1;
if (a > b && a > c)
{
status = 1;
}
else if (b > c)
{
status = 2;
}
else
{
status = 3;
}
switch (status)
{
case 1:System.out.println("a is the greatest");
```

```
break;
case 2:System.out.println("b is the
greatest");break;
case 3:System.out.println("c is the
greatest");break;
default:System.out.println("Cannot be determined");
}
}
}
```

Iteration statements

Iteration statements execute a block of code for several numbers of times until the condition is true.

While Statement

The while statement is one of the looping constructs control statement that executes a block of code while a condition is true. The loop will stop the execution if the testing expression evaluates to false. The loop condition must be a boolean expression. The syntax of the while loop is

```
while (<loop condition>)
<statements>
```

The following program explains the while statement.

```
public class ProgramWhile
{
public static void main(String[] args)
{
int count = 1;
System.out.println("Printing Numbers from 1 to
10");while (count <= 10)
{
System.out.println(count++);}
}
}
}
```

Do-while Loop Statement

The do-while loop is similar to the while loop, except that the test condition is performed at the end of the loop instead of at the beginning. The do—while loop executes atleast once without checking the condition.

It begins with the keyword do, followed by the statements that making up the body of the loop. Finally, the keyword while and the test expression completes the do-while loop. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop.

The syntax of the do-while loop is

```
do
<loop body>
while (<loop condition>);
```

The following program explains the do--while statement.

```
public class DoWhileLoopDemo {
public static void main(String[] args)
{int count = 1;
System.out.println("Printing Numbers from 1 to
10");do {
System.out.println(count++);
} while (count <= 10);
}
}
```

For Loop

The for loop is a looping construct which can execute a set of instructions for a specified number of times. It's a counter controlled loop.

The syntax of the loop is as follows:

```
for (<initialization>; <loop condition>; <increment expression>)
<loop body>
```

- initialization statement executes once before the loop begins. The <initialization> section can also be a comma-separated list of expression statements.
- test expression. As long as the expression is true, the loop will continue. If this expression is evaluated as false the first time, the loop will never be executed.

- Increment(Update) expression that automatically executes after each repetition of the loop body.
- All the sections in the for-header are optional. Any one of them can be left empty, but the two semicolons are mandatory.

The following program explains the for statement.

```
public class ProgramFor
{
public static void main(String[] args)
{
System.out.println("Printing Numbers from 1 to10");
for (int count = 1; count <= 10; count++)
{
System.out.println(count);
}
}
}
```

Transfer statements

Transfer statements are used to transfer the flow of execution from one statement to another.

Continue Statement

A continue statement stops the current iteration of a loop (while, do or for) and causes execution to resume at the top of the nearest enclosing loop. The continue statement can be used when you do not want to execute the remaining statements in the loop, but you do not want to exit the loop itself.

The syntax of the continue statement is

```
continue; // the unlabeled form
continue <label>; // the labeled form
```

It is possible to use a loop with a label and then use the label in the continue statement. The label name is optional, and is usually only used when you wish to return to the outermost loop in a series of nested loops.

The following program explains the continue statement.

```
public class ProgramContinue
{
public static void main(String[] args)
{System.out.println("Odd Numbers");
```

```
for (int i = 1; i <= 10; ++i) {
if (i % 2 == 0)
continue;
System.out.println(i + "\t");
}
}
}
```

Break Statement

The break statement terminates the enclosing loop (for, while, do or switch statement). Break statement can be used when we want to jump immediately to the statement following the enclosing control structure. As continue statement, can also provide a loop with a label, and then use the label in break statement. The label name is optional, and is usually only used when you wish to terminate the outermost loop in a series of nested loops.

The Syntax for break statement is as shown below;

```
break; // the unlabeled form break
<label>; // the labeled form
```

The following program explains the break statement.

```
public class ProgramBreak {
public static void main(String[] args) {
System.out.println("Numbers 1 - 10");
for (int i = 1;; ++i) {
if (i == 11)
break;
// Rest of loop body skipped when i is even
System.out.println(i + "\t");
}
}
}
```

The transferred statements such as try-catch-finally, throw will be explained in the later chapters.