

JAVA Programming Language

Data Types and operators



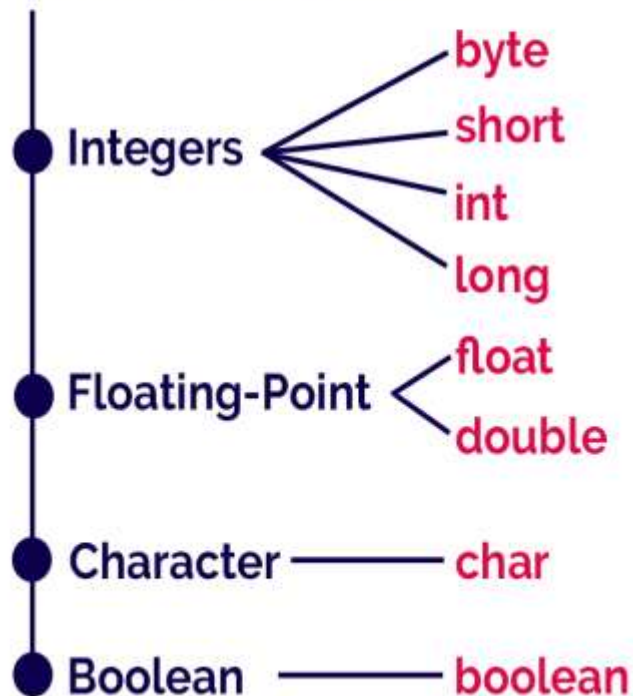
Java Data Types

- Data are used to specify what kind of value can be stored in a variable.
- The memory size and type of the value of a variable are determined by the variable data type..
- Data types are classified into two types:
 - **Primitive Data Types**
 - **Non-primitive Data Types**

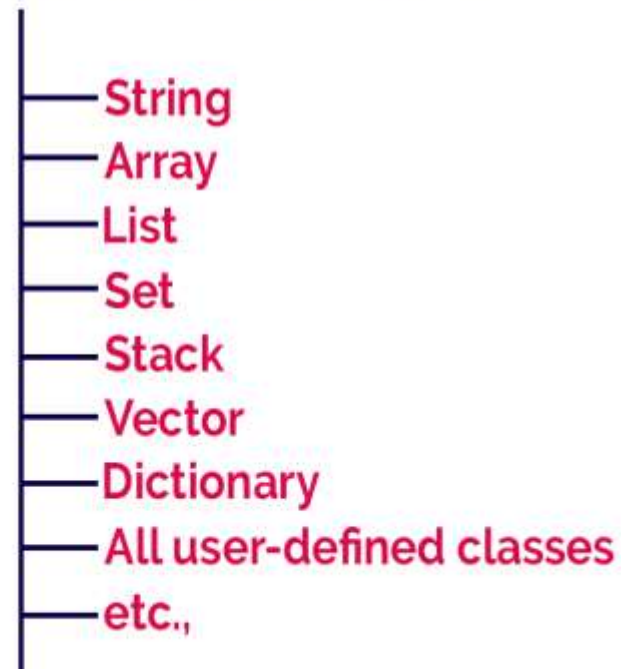


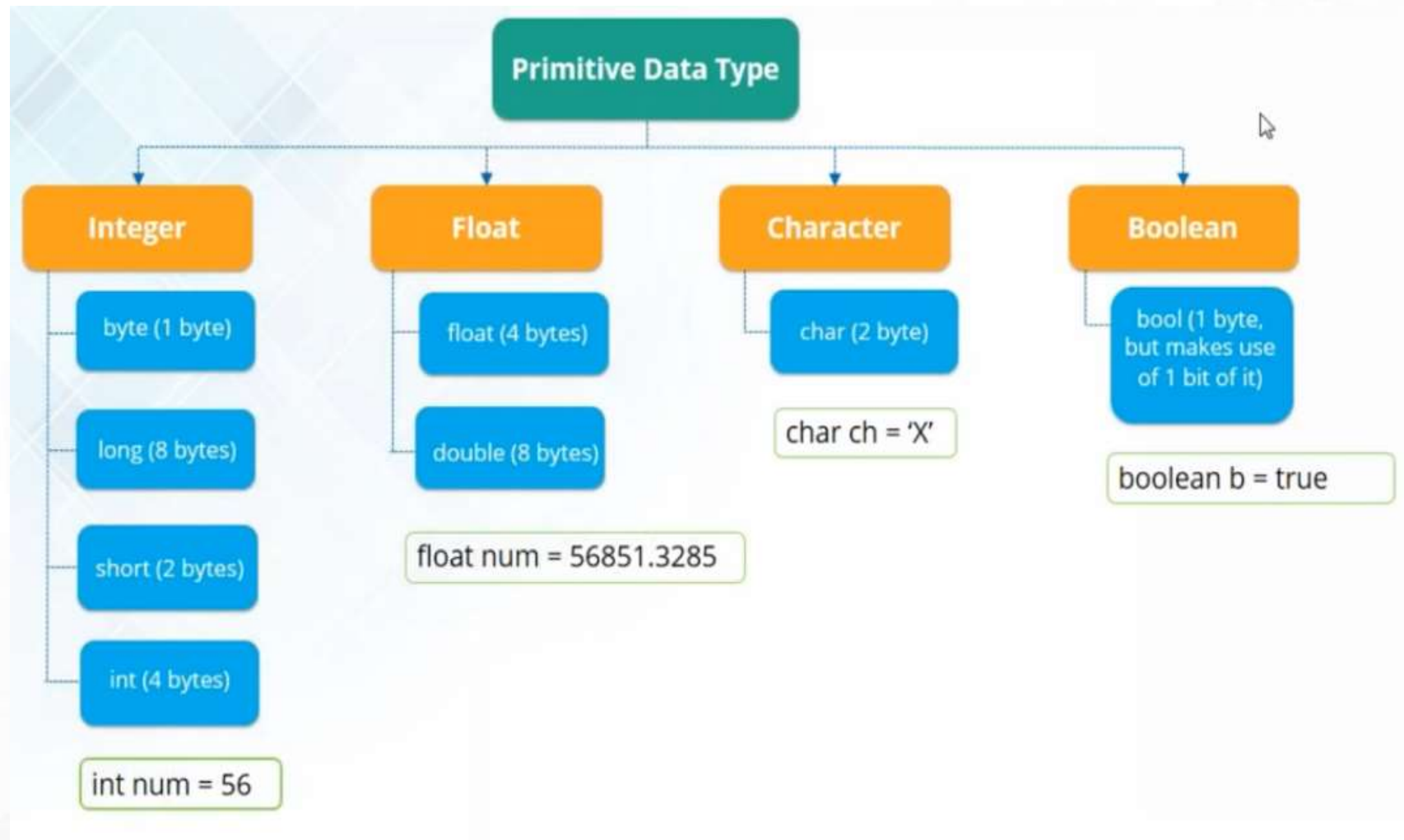
Data Types in java

Primitive Data Types



Non-primitive Data Types





John owns a Retail Department store. John needs to create a bill with the following fields present:

- Invoice ID
- Product ID
- Product Cost
- Quantity
- Discount
- Total Price
- Feedback Provided ?

John has no knowledge about the data types in java,

Can you help John ?



John owns a Retail Department store. John needs to create a bill with the following fields present:

- Invoice ID - Integer
- Product ID - Integer
- Product Cost - Double
- Quantity - Integer
- Discount - Double
- Total Price - Double
- Feedback Provided ? - Boolean



```
public class retail_shop {
```

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

```
        int invoice_num, product_id, Quant;
```

```
        double cost, discount, price;
```

```
        boolean feedback;
```

```
    }
```

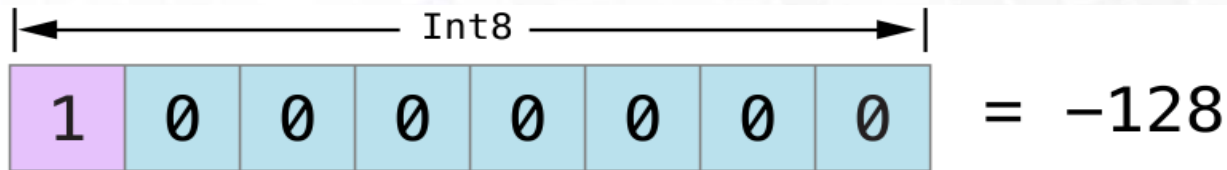
```
}
```



TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	-6,23,56,0	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\', '\n', '\t', '\r', '\f', '\b', '\e', '\a', '\c', '\d', '\f', '\n', '\r', '\t', '\v', '\x', '\u', '\U', '\e', '\a', '\c', '\d', '\f', '\n', '\r', '\t', '\v', '\x', '\u', '\U'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	-300,-4,45,100	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	3.40282347e+38 to 1.40239846e-45 (Approx.)
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	1.7976931348623157 e+308 to 4.9406564584124654 e-324 (Approx.)



- The Range of data, that can be stored in a variable is based on the memory size of the data type.
- For example: byte size is 1 byte(8 bits) and range is -128 to 127.
 - byte c;



Sign
bit

Value
bits

two's complement
representation



Example5: Java Data types – Default values

```
public class PrimitiveDataTypes
```

```
{    byte i;
```

```
    short j;
```

```
    int k;
```

```
    long l;
```

```
    float m;
```

```
    double n;        char ch;    boolean p;df
```

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

```
        PrimitiveDataTypes obj = new PrimitiveDataTypes();
```

```
        System.out.println("i = " + obj.i + ", j = " + obj.j + ", k = " + obj.k + ", l = " +  
obj.l);
```

```
        System.out.println("m = " + obj.m + ", n = " + obj.n);
```

```
        System.out.println("ch = " + obj.ch);
```

```
        System.out.println("p = " + obj.p);
```

```
    } }
```



Type Casting

- Programming is playing around with data. In [Java](#), there are many data types. Most of the times while coding, it is necessary to change the type of data to understand the processing of a variable and this is called Type Casting. In this article, I will talk about the fundamentals of Type Casting in Java.



Type Casting

- The process of converting the value of one data type (int, float, double, etc.) to another data type is known as typecasting.
- The automatic conversion is done by the compiler and manual conversion performed by the programmer.
 - There are two types of primitive type casting:
 - Widening Type Casting(automatic)
 - Narrowing Type Casting(manual)



Widening Casting

- This type of casting takes place when two data types are automatically converted. It is also known as Implicit Conversion. This happens when the two data types are compatible and also when we assign the value of a smaller data type to a larger data type.
- For Example, The numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other. Now let's write a logic for Implicit type casting to understand how it works.



Widening Type Casting(automatic)

- Widening Casting (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double
- Widening casting is done automatically when passing a smaller size type to a larger size type:
- Example:

```
public class Main { public static void main(String[] args)
{
int myInt = 9;
double myDouble = myInt; // Automatic casting: int to double
System.out.println(myInt); // Outputs 9
System.out.println(myDouble); // Outputs 9.0
}
}
```



Widening Type Casting(automatic)

- The lower data type (having smaller size) is converted into the higher data type (having larger size). Hence there is no loss in data. This is why this type of conversion happens automatically.
- **Note:** This is also known as **Implicit Type Casting**.



Narrowing Type Casting(manual)

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte
- Narrowing casting must be done manually by placing the type in parentheses in front of the value
- If we don't type cast manually the compiler will show an error

public class Main

```
{  
public static void main(String[] args)  
{  
    double myDouble = 9.78;  
    int myInt = (int) myDouble; // Manual casting: double to int  
    System.out.println(myDouble); // Outputs 9.78  
    System.out.println(myInt); // Outputs 9  
}  
}
```



Narrowing Type Casting(manual)

- The higher data types (having larger size) are converted into lower data types (having smaller size). Hence there is the loss of data. This is why this type of conversion does not happen automatically.
- **Note:** This is also known as **Explicit Type Casting**.



Java Operators



Operators

- Operators are used to perform a specific kind of operations on variables and values.
- Classification of java Operators:
 - If an operator takes one operand, it is **unary operator**; (++,--,!,~,+,-)
 - if an operator takes two operands, it is **binary operator**; (*,%,/,+,>,<)
 - If an operator takes three operands, it is **ternary operator**. (?:)



Data Operations

Arithmetic Operators

+ → Addition
- → Subtraction
* → Multiplication
/ → Division
% → Modulus

Unary operators

++ → Increment operator
-- → Decrement operator

Relational operators

< → Less than
<= → Less than or equal to
> → Greater than
>= → Greater than or equal to
!= → Not equal to
== → Equals

Logical Operators

&& → And
|| → Or
! → Not



Let us assume the following values for the variables:

- Num1 = 10
- Num2 = 15
- Num3 = 25
- Num4 = true

Operation	Solution
$(\text{Num1} + \text{Num2}) - \text{Num3} / 5$	20
Num1++	11
$\text{Num3} \geq \text{Num1} + \text{Num2}$	True
$\text{Num2} == \text{Num1}$	False
$(\text{Num2} < \text{Num3}) \ \&\& \ (\text{Num1} > \text{Num3})$	False
!Num4	False



Operator Type	Example
Unary	<code>++ -- + - ! ~</code>
Arithmetic	<code>+ - * / %</code>
Relational / Conditional	<code>< > <= >= == !=</code>
Logical	<code>& ^ ! && </code>
Bit – wise Operators	<code>& ^ ~ << >> >>></code>
Ternary	<code>?:</code>
Assignment	<code>= += -= *= /= %= &= ^= = <<= >>= >>>=</code>



Mathew has come to John's store to purchase a few products. Below are the items Mathew wishes to purchase :

Product	Quantity
Item A	2
Item B	1
Item C	3

Below are the cost of the products:

Product	Cost
Item A	200
Item B	75
Item C	500

At the time Mathew visited the store, there was a 10% discount on all product.

Finally a service tax of 5% is applicable on all products.

Can you help John compute the amount Mathew has to pay ?



```
public class retail_shop {  
  
    public static void main(String[] args) {  
        int iteama=200; int iteamb = 75; int iteamc = 500;  
        double price ;  
  
        price=((iteama*2)+(iteamb)+(iteamc*3));  
  
        price= price-(.1*price);  
  
        price= price+(.05*price);  
  
        System.out.println(price);  
  
    }  
}
```



Unary Operator

- The Java unary operators require only one operand.
- Example:

```
public class Main{  
    public static void main(String[] args){  
        int a=10,b=20;  
        boolean ch=false;  
        System.out.println(a++);  
        System.out.println(a);  
        System.out.println(++b);  
        a=-b;  
        System.out.println(a);  
        a=~16;  
        System.out.println(a);  
        System.out.println(!ch);  
    }  
}
```

```
10  
11  
21  
-21  
-17  
true
```



Arithmetic Operators

Arithmetic operators are used to performing basic mathematical operations

Operator	Meaning	Example
+	Addition	$10 + 5 = 15$
-	Subtraction	$10 - 5 = 5$
*	Multiplication	$10 * 5 = 50$
/	Division	$10 / 5 = 2$
%	Modulus - Remainder of the Division	$5 \% 2 = 1$
++	Increment	a++
--	Decrement	a--



Arithmetic Operators

- The addition operator can be used with numerical data types and character or string data type. When it is used with numerical values, it performs mathematical addition and when it is used with character or string data type values, it performs concatenation (appending).
- The modulus (remainder of the division) operator is used with integer data type only.
- The **increment** and **decrement** operators are used as **pre-increment** or **pre-decrement** and **post-increment** or **post-decrement**.
 - 🔔 When they are used as pre, the value is get modified before it is used in the actual expression and when it is used as post, the value is get modified after the actual expression evaluation.



Example6- Arithmetic Operators

```
public class ArithmeticOperators {  
  
    public static void main(String[] args) {  
        int a = 10, b = 20, result;  
        System.out.println("a = " + a + ", b = " + b);  
  
        result = a + b;  
        System.out.println("Addition: " + a + " + " + b + " = " + result);  
  
        result = a - b;  
        System.out.println("Subtraction: " + a + " - " + b + " = " + result);  
  
        result = a * b;  
        System.out.println("Multiplucation: " + a + " * " + b + " = " + result);  
  
        result = b / a;  
        System.out.println("Division: " + b + " / " + a + " = " + result);  
  
        result = b % a;  
        System.out.println("Modulus: " + b + " % " + a + " = " + result);  
  
        result = ++a;  
        System.out.println("Pre-increment: ++a = " + result);  
  
        result = b--;  
        System.out.println("Post-decrement: b-- = " + result);  
    }  
}
```



Relational Operators

- The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values.
- Relational operator has two possible results either **TRUE** or **FALSE**.
- The relational operators are used to define conditions in a program.



Relational Operators

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE



Example7- Relational Operators

```
public class RelationalOperators {  
  
    public static void main(String[] args) {  
  
        boolean a;  
  
        a = 10<5;  
        System.out.println("10 < 5 is " + a);  
  
        a = 10>5;  
        System.out.println("10 > 5 is " + a);  
  
        a = 10<=5;  
        System.out.println("10 <= 5 is " + a);  
  
        a = 10>=5;  
        System.out.println("10 >= 5 is " + a);  
  
        a = 10==5;  
        System.out.println("10 == 5 is " + a);  
  
        a = 10!=5;  
        System.out.println("10 != 5 is " + a);  
    }  
}
```



Logical Operators

- The logical operators are the symbols that are used to combine multiple conditions into one condition.

Operator	Meaning	Example
&	Logical AND - Returns TRUE if all conditions are TRUE otherwise returns FALSE	false & true => false
	Logical OR - Returns FALSE if all conditions are FALSE otherwise returns TRUE	false true => true
^	Logical XOR - Returns FALSE if all conditions are same otherwise returns TRUE	true ^ true => false
!	Logical NOT - Returns TRUE if condition is FALSE and returns FALSE if it is TRUE	!false => true
&&	short-circuit AND - Similar to Logical AND (&), but once a decision is finalized it does not evaluate remaining.	false & true => false
	short-circuit OR - Similar to Logical OR (), but once a decision is finalized it does not evaluate remaining.	false true => true



Logical Operators

- The operators **&**, **|**, and **^** can be used with both boolean and integer data type values. When they are used with integers, performs bitwise operations and with boolean, performs logical operations.
- **Logical operators** and **Short-circuit operators** both are similar, but in case of short-circuit operators once the decision is finalized it does not evaluate remaining expressions.



Example 8- Logical operators

```
public class LogicalOperators {  
  
    public static void main(String[] args) {  
  
        int x = 10, y = 20, z = 0;  
        boolean a = true;  
  
        a = x>y && (z=x+y)>15;  
        System.out.println("a = " + a + ", and z = " + z);  
  
        a = x>y & (z=x+y)>15;  
        System.out.println("a = " + a + ", and z = " + z);  
  
    }  
  
}
```

```
a = false, and z = 0  
a = false, and z = 30
```



Assignment Operators

- The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue).

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10
--	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -- B
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B
&=	Logical AND assignment	-
=	Logical OR assignment	-
^=	Logical XOR assignment	-



Example 9- Assignment Operators

```
public class AssignmentOperators {  
  
    public static void main(String[] args) {  
  
        int a = 10, b = 20, c;  
        boolean x = true;  
        System.out.println("a = " + a + ", b = " + b);  
        a += b;  
        System.out.println("a = " + a);  
        a -= b;  
        System.out.println("a = " + a);  
        a *= b;  
        System.out.println("a = " + a);  
        a /= b;  
        System.out.println("a = " + a);  
        a %= b;  
        System.out.println("a = " + a);  
        x |= (a>b);  
        System.out.println("x = " + x);  
        x &= (a>b);  
        System.out.println("x = " + x);  
    }  
}
```



Bitwise Operators

- The bitwise operators are used to perform bit-level operations in the java programming language.
- In the bitwise operators, the operations are performed based on binary values.



Bitwise Operators

Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)



Example 10 - Bitwise Operators

```
public class BitwiseOperators {  
  
    public static void main(String[] args) {  
  
        int a = 25, 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);  
  
        System.out.println(a + ">>" + 2 + " = " + (a>>2));  
  
        System.out.println(a + "<<" + 2 + " = " + (a<<2));  
  
        System.out.println(a + ">>>" + 2 + " = " + (a>>>2));  
  
    }  
}
```



Program to find a no ODD or EVEN

```
public class Main{  
    public static void main(String[] args)  
    {  
        int a=20;  
        if((a&1)==0)  
            System.out.println("Even NO");  
        else  
            System.out.println("ODD NO");  
    }  
}
```



```
public class Main{  
    public static void main(String[] args)  
    {  
        int a=20;  
        System.out.println("a * 2 is:"+(a<<1));  
        System.out.println("a / 2 is:"+(a>>1));  
        System.out.println("a / 8 is:"+(a>>3));  
    }  
}
```



To check a no is powers of 2 or not

```
public class Main{  
    public static void main(String[] args)    {  
        int a=128;  
        if((a&a-1)==0)  
            System.out.println(a+" is power of 2 ");  
        else  
            System.out.println(a+" is not power of 2");  
    }  
}
```



Bitwise Operators - Signed right shift operator(>>) vs Un - Signed right shift operator(>>>)

- The signed right shift operator '>>' uses the sign bit to fill the trailing positions.
- The unsigned right shift operator '>>>' do not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.
- Assume if a = 60 and b = -60; now in binary format, they will be as follows –
a = 0000 0000 0000 0000 0000 0000 0011 1100 (60)
b = 1111 1111 1111 1111 1111 1111 1100 0100 (-60)
→ In Java, negative numbers are stored as 2's complement.

a >> 1 = 0000 0000 0000 0000 0000 0000 0001 1110 (30)

b >> 1 = 1111 1111 1111 1111 1111 1111 1110 0010 (-30)

a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110 (30)

b >>> 1 = 0111 1111 1111 1111 1111 1111 1110 0010 (2147483618)



Bitwise Operators - Signed right shift operator(>>) vs Un - Signed right shift operator(>>>)

```
public class BitwiseOperators {  
  
    public static void main(String[] args) {  
  
        int a = 60, b = -60;  
        System.out.println(a + ">>" + 2 + " = " + (a>>1));  
        System.out.println(a + ">>" + 2 + " = " + (b>>1));  
        System.out.println(a + ">>>" + 2 + " = " + (a>>>1));  
        System.out.println(a + ">>>" + 2 + " = " + (b>>>1));  
    }  
}
```



Conditional Operators / Ternary Operator

- `?:` \rightarrow is called a **ternary operator** because it requires three operands.
- First, It will verify a condition, then perform one operation out of the two operations based on the condition result.
- If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed.
- **Syntax**

Condition ? TRUE Part : FALSE Part;



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



Java expressions

- An expression is a construct which is made up of literals, variables, method calls and operators

Expression evaluation in Java is based upon the following concepts:

- Type promotion rules (Type casting)
- Operator precedence
- Associativity rules
- Ex: `int a=34+67*12%34&(23+67)/34^56;`



```
public class ConditionalOperator {  
    public static void main(String[] args) {  
        int a = 50, b = 20, c;  
        double d;  
        System.out.println(a++);  
        System.out.println(++b);  
        d=a/b;  
        System.out.println(d);  
        d=(double)a/b;  
        System.out.println(d);  
        c=a+b*(34/56-23*b+a);  
        System.out.println(c);  
    }  
}
```



Java Operator Precedence Table

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5		Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3		Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= % =	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.



Thank you




Decision making in java.

- Consider a real life example, you are driving a car on a road to a concert, and you see your fuel light flashing, you immediately decide that you will stop at the next petrol pump. But if the light would not have blinked at all, you would have continued driving. Right? That is where you took a decision and planned your action.



Decision Making in Java



An illustration on the left side of the slide shows a man in a blue shirt and red pants standing on a yellow ladder, pointing at a large black screen filled with colorful lines of code. A woman in a yellow dress stands next to him, also pointing at the screen. The background is a dark teal color with a subtle grid pattern.

Decision Making in Java

01	if	04	if-else-if
02	if-else	05	switch-case
03	nested-if	06	jump-break, continue, return

Definition of Decision making in java.

- Decision making in Java executes a particular segment of code based on the result of a boolean condition. It is important because conditions define the flow of programs and the output of a particular program.
- The decision making principles in Java chiefly consist of if else statements, continue, break and switch statements. It decides the flow of the program control during the execution of the program.



There are the 6 ways of exercising decision making in Java:

- 1. if
- 2. if-else
- 3. nested-if
- 4. if-else-if
- 5. switch-case
- 6. jump-break, continue, return

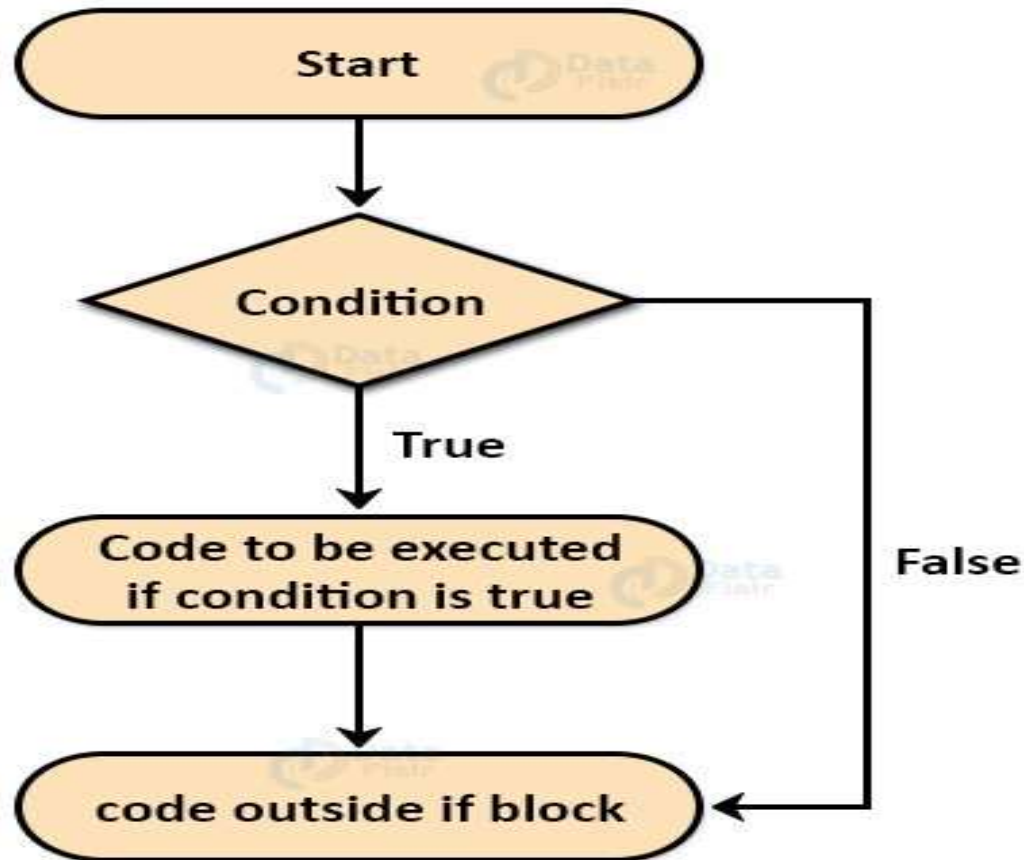


1.if

- Java if statement is the **simplest decision making statement**. It encompasses a boolean condition followed by a scope of code which is executed only when the condition evaluates to true. However if there are no curly braces to limit the scope of sentences to be executed if the condition evaluates to true, then only the first line is executed.



Flow diagram for if statement



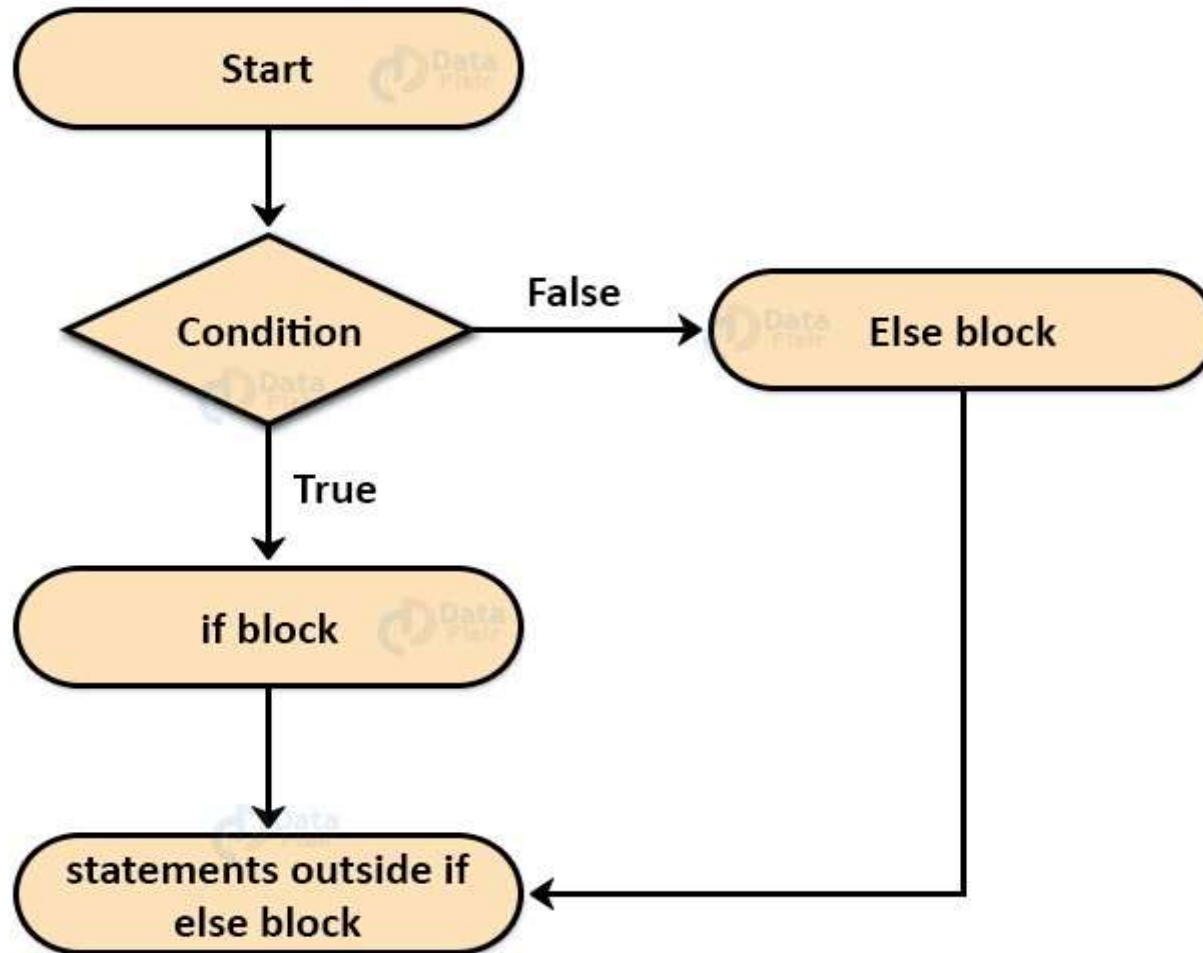
2.If-else

- This pair of keywords is used to divide a program to be executed into two parts, one being the code to be executed if the condition evaluates to true and the other one to be executed if the value is false.
- However if no curly braces are given the first statement after the if or else keyword is executed.

- **if(condition)**
 {
 //code to be executed if the condition is true
 }
 else
 {
 //code to be executed if the condition is false
 }



Flow diagram of if-else statement



3. Nested if Statements in Java

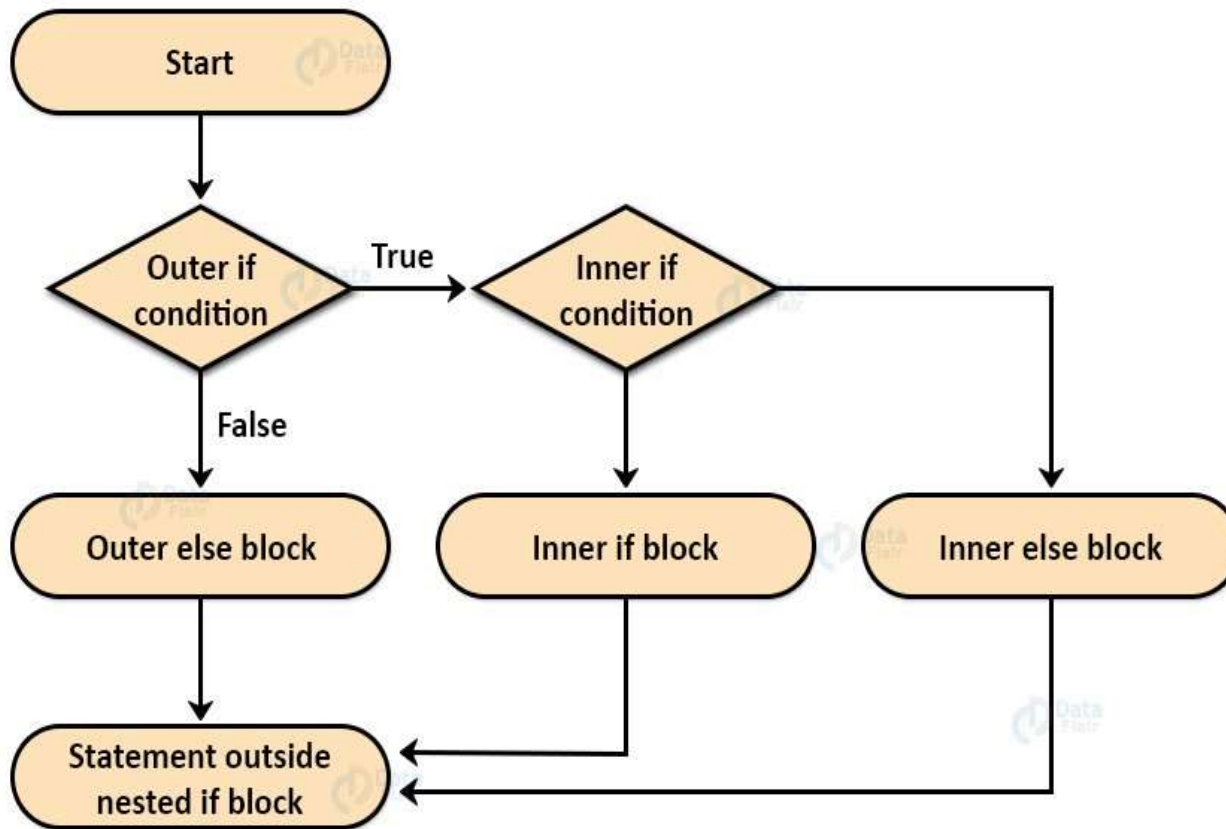
- The nested if is similar to the nested loops we learnt about in the previous chapters. If the condition of the outer if statement evaluates to true then the inner if statement is evaluated.

Nested if's are important if we have to declare extended conditions to a previous condition

- Syntax:
- ```
if(condition)
{
//code to be executed
if(condition)
{
//code to be executed
}
}
```



## Flow diagram a nested if statement



## 4. if-else-if Statements in Java

- These statements are similar to the if else statements . The only difference lies in the fact that each of the else statements can be paired with a different if condition statement. This renders the ladder as a multiple choice option for the user. As soon as one of the if conditions evaluates to true the equivalent code is executed and the rest of the ladder is ignored.

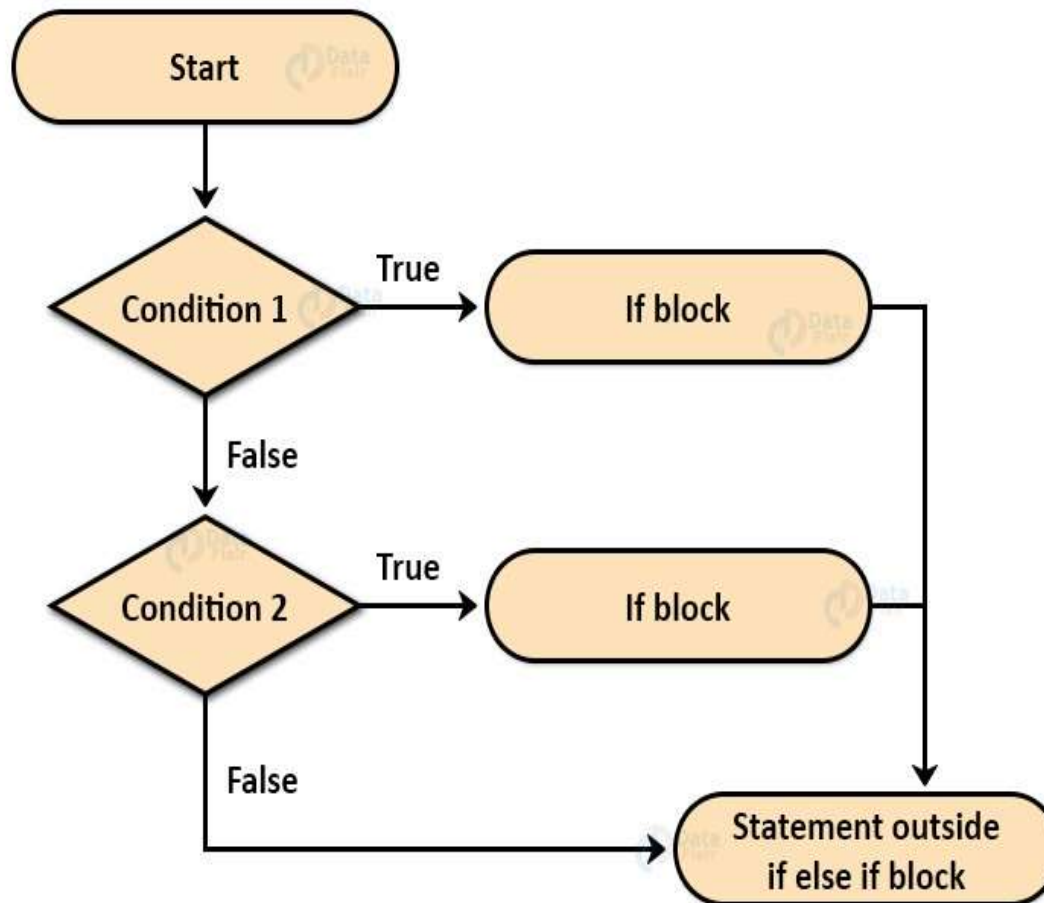


# Syntax for

- **Syntax:**
- **if**  
**{**  
**//code to be executed**  
**}**  
**else if(condition)**  
**{**  
**//code to be executed**  
**}**  
**else if(condition)**  
**{**  
**//code to be executed**  
**}**  
**else**  
**{**  
**//code to be executed**  
**}**



## Flow Diagram of if else if statements





## 5. Switch Statement in Java

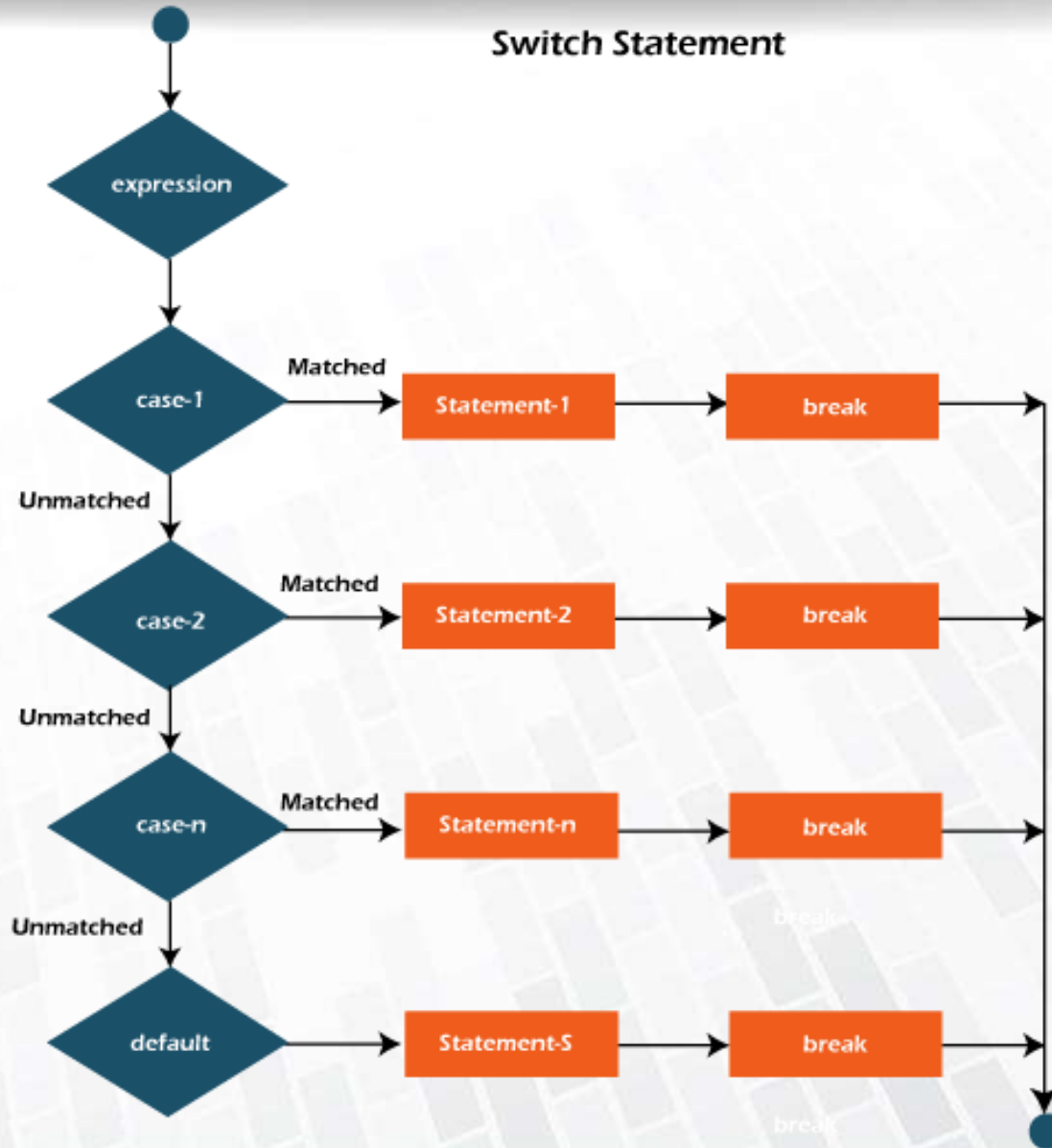
- Java switch statement is a different form of the if else if ladder statements.
- This saves the hassle of writing else if for every different option.
- It branches the flow of the program to multiple points as and when required or specified by the conditions.
- It bases the flow of the program based on the output of the expression.
- As soon as the expression is evaluated, the result is matched with each and every case listed in the scope. If the output matches with any of the cases mentioned, then the particular block is executed. A break statement is written after every end of the case block so that the remaining statements are not executed.
- The default case is written which is executed if none of the cases are the result of the expression. This is generally the block where error statements are written.



- `switch(expression)`  
  `{`  
    `case <value1>:`  
      `//code to be executed`  
      `break;`  
    `case <value2>:`  
      `//code to be executed`  
      `break;`  
    `default:`  
      `//code to be defaultly executed`  
  `}`



## Switch Statement



## 6. Jump Statements in Java

- The jump statements are the keywords in Java which have specific meaning and **specific action of disrupting the normal flow** of the program.

### Types of Jump Statements

- Break statement
  - Break
  - Break label
- The continue statement
- The return statement

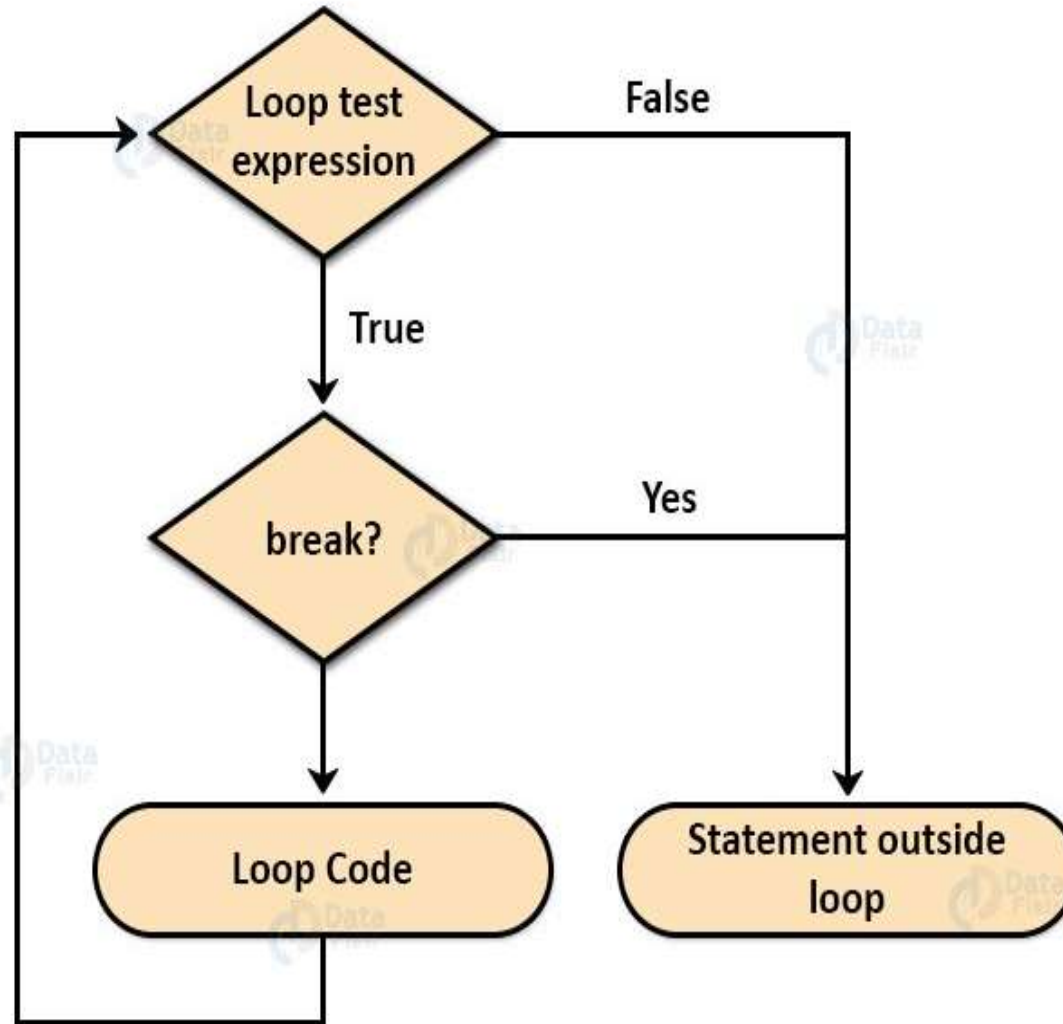


## a. Java break statement

- The break statement , as the name signifies, is useful to break out of the normal workflow of a program. We can use it to **terminate loops**, end cases of **switch statements**, and as a goto statement. Break statements, if used in a nested loop, terminates the innermost loop. The outermost loop still functions.



## Flowchart of break statement

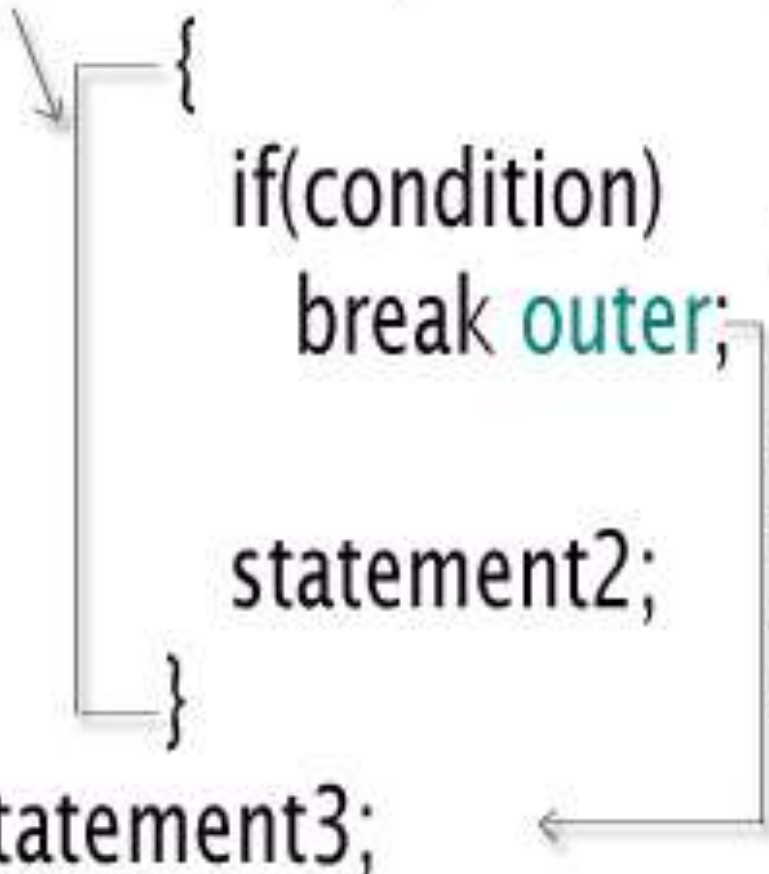




## b. break label statement

```
outer: while(condition)
```

```
{
 if(condition)
 break outer;
 statement2;
}
statement3;
```



while loop is labelled as "outer" and hence this statement "break outer" breaks the control out of the loop named "outer", without executing statement2.





```
int i=0;
loop1: while(i<20) {
 if(i==10)
 break loop1;
 System.out.println(i);
 i++;
}
```



## b. Continue Statement in Java

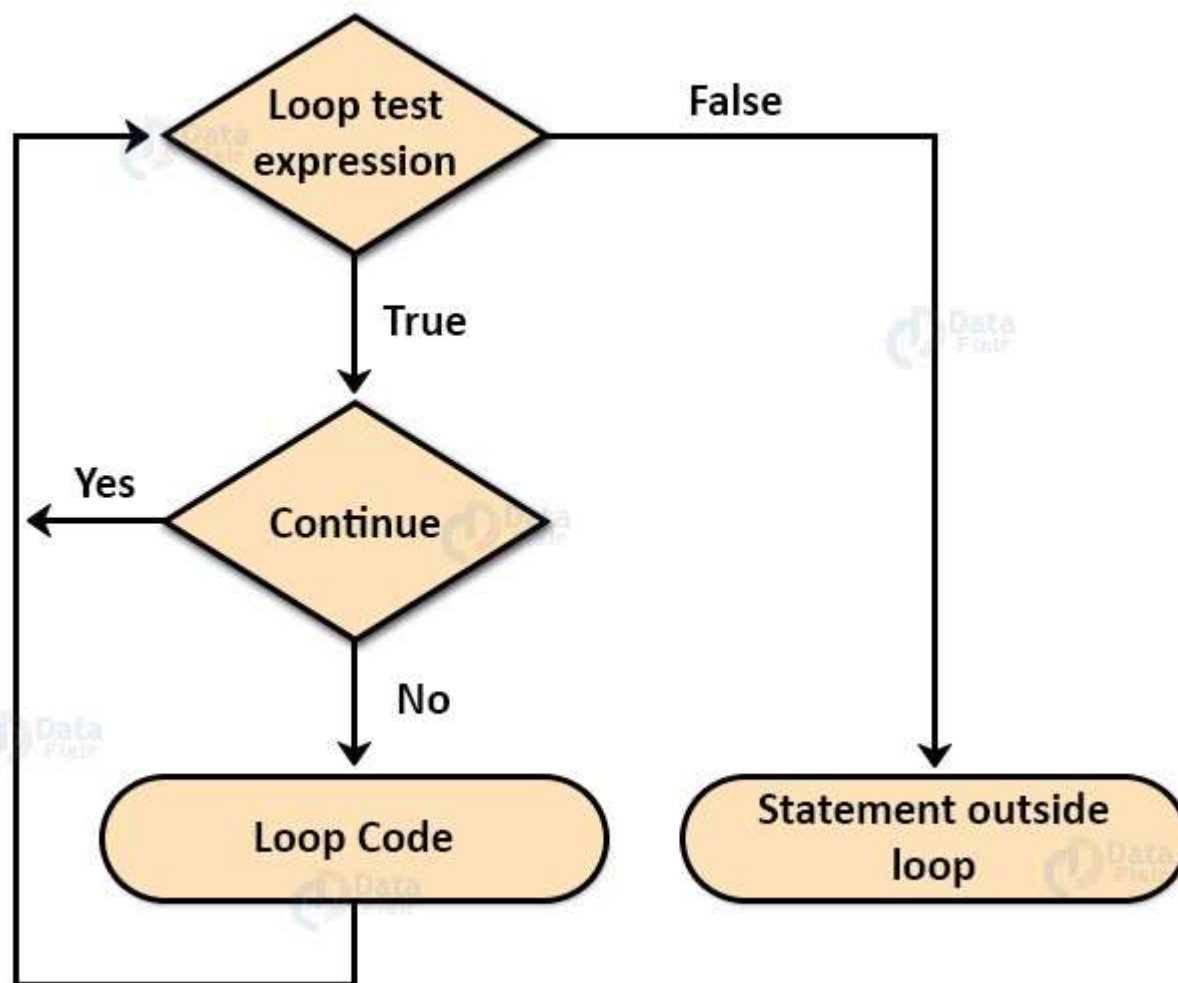
- The continue statement is useful for early iteration of a particular loop. Sometimes rather than breaking out of a loop and halting it for good we want to skip some iterations based on our requirements of a program. This results in the usage of continue statements.



```
class Main {
 public static void main(String args[]) {
 int i;
 for(i=1;i<=10;i++) {
 if(i==5) continue;
 { System.out.print(i + " "); }
 }
 }
}
```



## Flow Diagram for the continue statement



# The Return Statement

- It used for returning some value
- It specified at the end of method

