CHAPTER 3 JAVA GUI PROGRAMMING REVISION TOUR - I

TOKENS

The smallest individual unit a program is known as a **Token**. Java has following types of tokens:

- Keywords
- Identifiers
- Literals
- Punctuators/Separators
- Operators

KEYWORDS

- These are those words that convey a special meaning to language compiler. These are reserved for special purpose and **must not be used as a identifier name.**
- e.g. int , float, double, char, String , break are some examples of keyword.

IDENTIFIERS

- These are the fundamental building block of a program and are used as the general terminology for the names given to different parts of the program namely, variables etc.

Identifiers forming rules of java:

1. Identifiers can have alphabets, digits and underscore and dollar sign characters.

2. They must not be a keyword.

3. They must not begin with a digit.

4. They can be of any length.

5. Java is case-sensitive i.e. upper-case and lower-case letters are treated differently.

Examples of some valid identifiers are:

myfile	$abc
abc_12	_abc

LITERALS

Also referred as constants, and are data items that are fixed data values. Literals available in Java are:

- Integer-literal
- Floating-literal
- Boolean-literal
- Character-literal
- String-literal
- Null-literal

Integer-Literal

- These are the whole numbers without any fractional part. **Rules of writing integer constants are:**

1. It must have at least one digit and must not contain any decimal points.
2. Commas cannot appear in an integer constant.

3. It may contain either + or – sign.

- Examples of valid integer literals are: 56 , +8902 , -235 , 090 etc.

- Java allows **three types of integer literals:**

    1. Decimal (base 10)

    2. Octal (base 8)

    3. HexaDecimal (base 16)

Floating-Literals (Real Literals)

- Real literals are having fractional parts. **Rules of writing floating literals are :**

1. It must have at least one digit before a decimal point and at least one digit after the decimal point.

2. Commas cannot appear in a Floating literal.

3. It may contain either + or – sign.

- **Examples of valid floating literals are**: 2.0 , +17.5 , -13.0 , -0.000875 etc.

- Real literals may be written in two forms: **FRACTIONAL FORM , EXPONENT FORM**

REAL LITERAL IN EXPONENT FORM

A real literal in exponent form consists of two parts: **mantissa and exponent.**
e.g.

5.8 can be written in exponent form as: $0.58 \times 10^1 = 0.58E1$

Where mantissa part is 0.58(the part appearing before E) and exponent part is 1 (the part appearing after E).

Boolean-literal

- A boolean literal is having two values only i.e. **true and false.**

Character-literal

- It is one character enclosed in single quotes, e.g. 'z' . **Rule of writing character literals are :**

1. It must contain one character and must be enclosed in single quotation marks.

e.g. 'C' , 'y' , '1' , '8' , '#' etc.

NONGRAPHIC CHARACTERS

- Those characters that cannot be typed directly from keyboard e.g. backspace, tabs etc.

- These characters are represented by using escape sequence. **An escape sequence is represented by a backslash followed by one or more characters.**

E.g.

\n – represents next line escape sequence.
\t – represents horizontal tab escape sequence.
\v – represents vertical tab escape sequence.

String-literals

- Multiple character constants are treated as string literal. **Rule of forming String-Literals are :**
  1. It is a sequence of zero or more characters surrounded by double quotes.
  e.g. "abc" , "raj" , "ko123" , "123" etc.

Null-literals

- A null literal is having value ***null*** only.

SEPARATORS

- The following **nine characters** are the separators :

( ) { } [ ] ; , .

OPERATORS

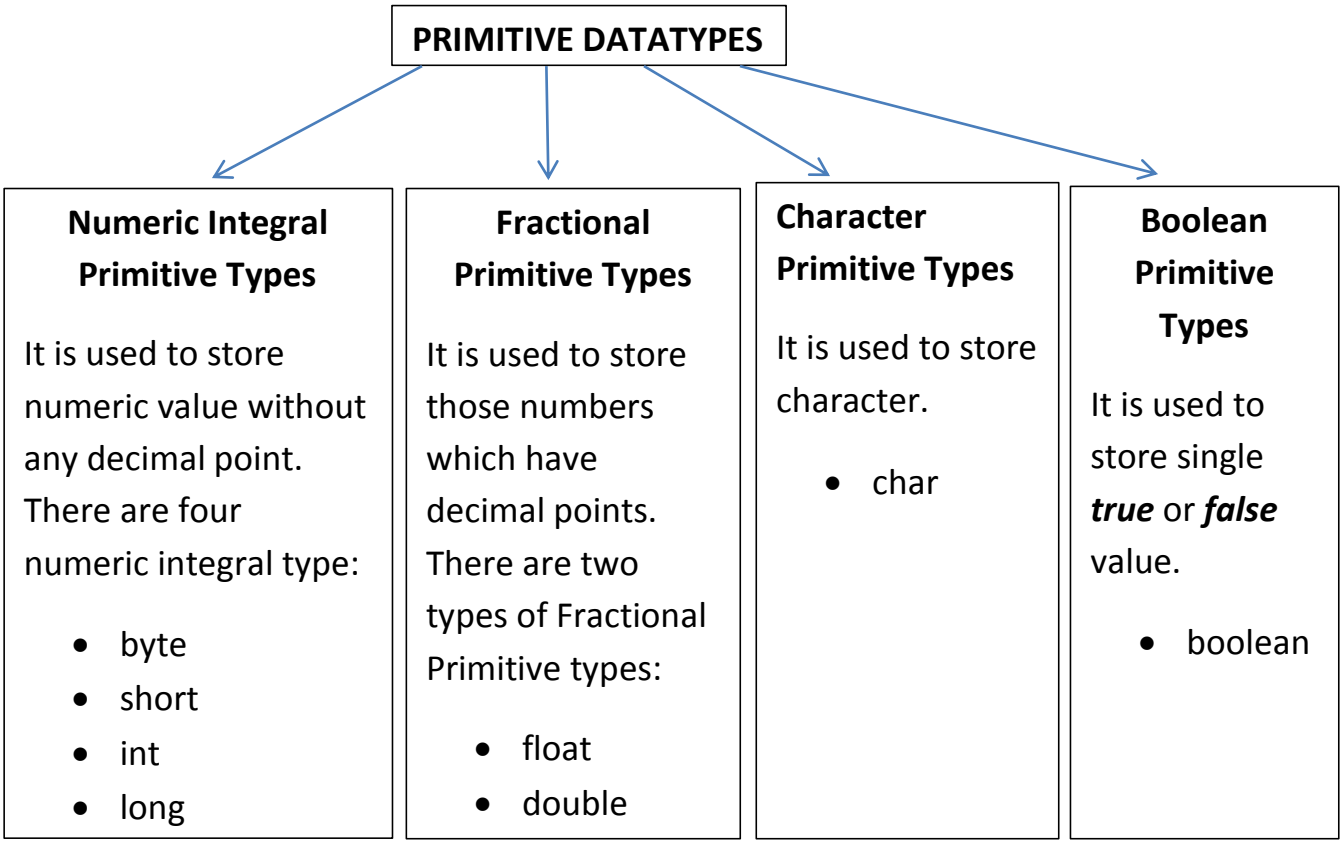
- Operator are those character that specify a particular operation.
- E.g. **+ , - , * , < , >** etc.
- Java has total **37 operators.**

DATA TYPES

- Data can be many types e.g. character, integer, real, string etc., and in order to handle different type of data, java provides Data types.
- Data types are means to identify the type of data and associated operations of handling it.
- Java data types are of **two types : PRIMITIVE DATA TYPE , REFERENCE DATA TYPES**

PRIMITIVE DATA TYPES (FUNDAMENTAL DATA TYPES)

- It comes as part of the language. Java provides eight primitive datatypes which are : byte, short , int, long,float, double, char, Boolean.



SIZE & RANGE OF EACH DATATYPE

TYPE	SIZE	RANGE
byte	8 bits (1 byte)	-128 to +127
short	16 bits (2 bytes)	-32768 to +32767
int	32 bits (4 bytes)	-2^{31} to $2^{31}-1$
long	64 bits (8 bytes)	-2^{63} to $2^{63}-1$
float	32 bits (4 bytes)	-3.4E+38 to +3.4E+38
double	64 bits (8 bytes)	-1.7E+308 to +1.7E+308
char	16 bits (2 bytes)	0 to 65536
boolean	1 bit	true or false

SOME DATA VALUES AND THEIR DATA TYPES

VALUE	DATATYPE
178	
8864	
37.26	
87.636	
'c'	
true	
false	

REFERENCE DATA TYPES

- These are constructed from primitive data types. **E.g. classes, array and interface**.

VARIABLES

- A variable is a named memory location, which holds a data value of a particular data type.

  e.g. the following statement declares a variable i of the data type int :

  int i ;

  Declare a variable **j** of **float** data type: _____

  Declare a variable **ch** of **char** data type:_____

DECLARATION OF VARIABLES

- The declaration of a variable generally takes the following form:

  type variablename ;

Any Valid Java Data type

It is the name of variable. A *variablename* is an identifier. Thus all rules of identifier naming apply to the name of a variable.

- E.g. Following declaration creates a variable ***age*** of ***int*** type :

  int age ;

- When **more than one variable of same data type** needs to declared then we can write the declaration as :

  int year, day , month ;

  double salary , wage ;

INITIALISATION OF VARIABLES

- All examples given above in declaration of variable, does not provide initial value or first value to variable.
- A variable with declared first value is said to be an initialised variable. E.g.

int age = 18 ;

double price = 2500.35 ;

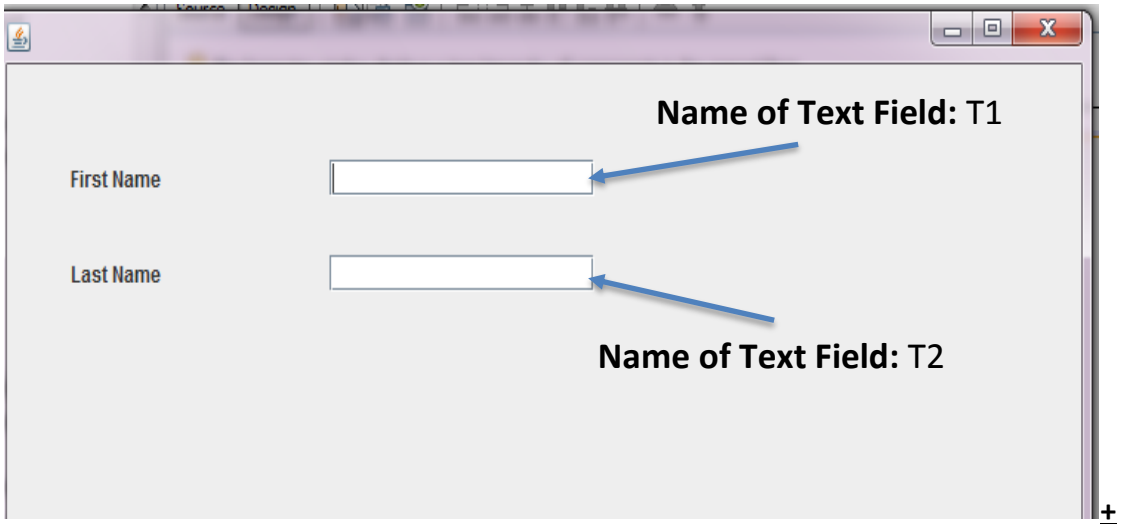
TEXT INTERACTION IN GUIs

- For text interaction in a GUI, **four types of methods are used :**
  1. getText( ) method
  2. parse……( ) method
  3. setText( ) method
  4. JOptionPane.showMessageDialog( ) method

getText( ) method – used to obtain text from a GUI component

- This method **returns the text currently stored in a text based GUI component**.
- Components that support getText( ) method include : TextField, Text Area, Button, Label, CheckBox, and RadioButton.

e.g. Consider the following GUI:



- To obtain text from T1 text field, we write : **T1.getText( ) ;**

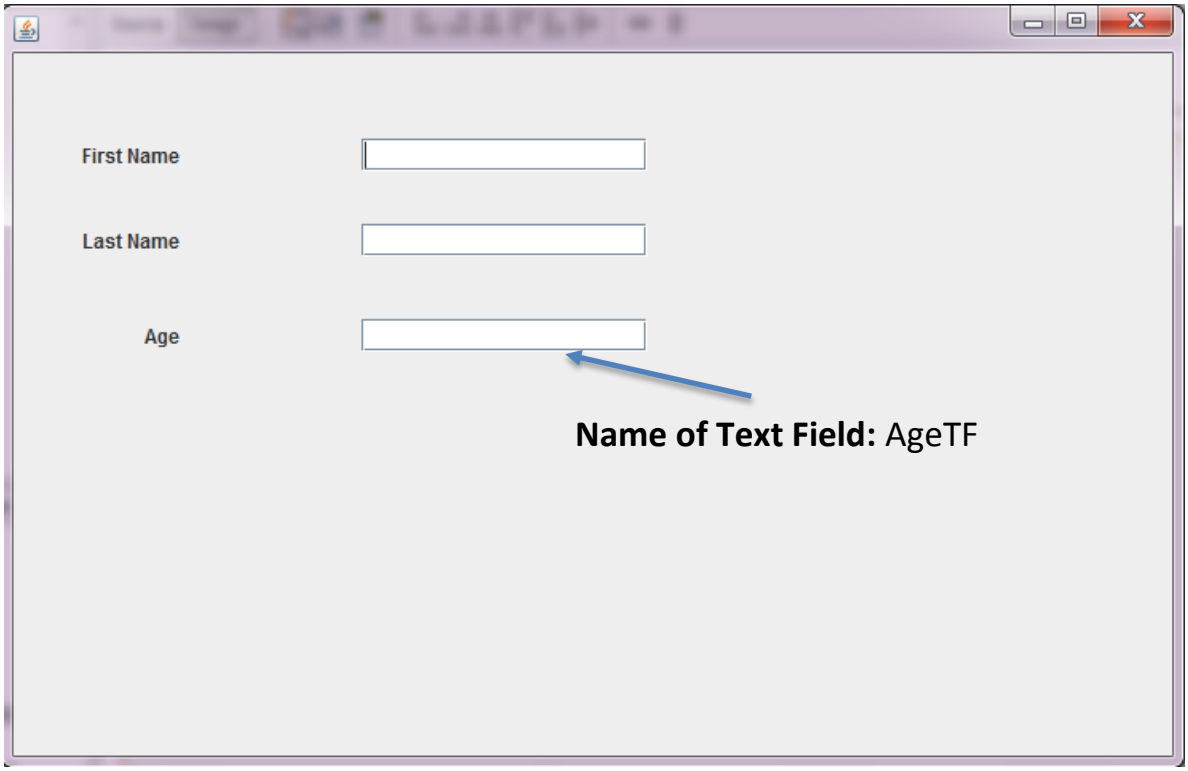
- The **getText( )** returns a value of **String** type, so we must store the value returned by getText( ) in String type variable. Thus complete statement to obtain text from T1 field would be :

  String str1 = T1.getText ( ) ;
- Similarly to obtain text from T2 textfield, we write :

  String Str2 = T2.getText( ) ;

parse……( ) methods – Obtaining numbers from a GUI component.

- Sometimes, we use text type components in a GUI but we intend to use it for obtaining numeric values e.g. we may want to read ***age*** of a person through a text field.



- **Since a text field will return text, i.e. String type of data**, you need a method that helps you extract/convert this textual data into a numeric type. For this, **parse( )** methods are useful.

- **There are many parse…….( ) methods that help you parse string into different numeric types. These are :**

(i) Byte.parseByte (String s) converts a String s into a byte type value.
(ii) Short.parseShort (String s) converts a String s into a short type value.
(iii) Integer.parseInt (String s) converts a String s into an int type value.
(iv) Long.parseLong (String s) converts a String s into a long type value.
(v) Float.parseFloat (String s) converts a String s into a float type value.
(vi) Double.parseDouble (String s) converts a String s into a double type value.

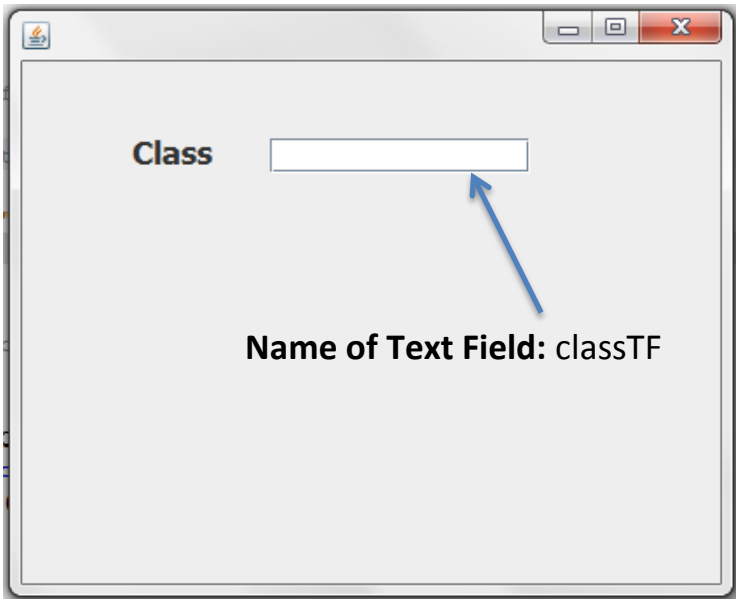
- Consider GUI given above, If we want to obtain input from AgeTF textfield, in numeric, say **int ,** form, we need to do it in two steps :
  - First, we have to obtain text from AgeTF by typing a statement like:
    String a = AgeTF.getText ( );
  - Then, we need to parse the **String a** obtained above into an **int** by typing statement like :
    int cl = Integer.parseInt (a);
- The above two steps can be combined into one also,

int cl = Integer.parseInt (AgeTF.getText( ));

- Similarly to obtain a float value, we may use
  Float.parseFloat(<text obtained from field>);
  To obtain a long value, we may use
  Long.parseLong(<text obtained from field>);

setText( ) method - Storing text into a GUI component

- This method changes text in a GUI component. The Swing components that support setText( ) method include : TextField, TextArea, Button, Label etc.



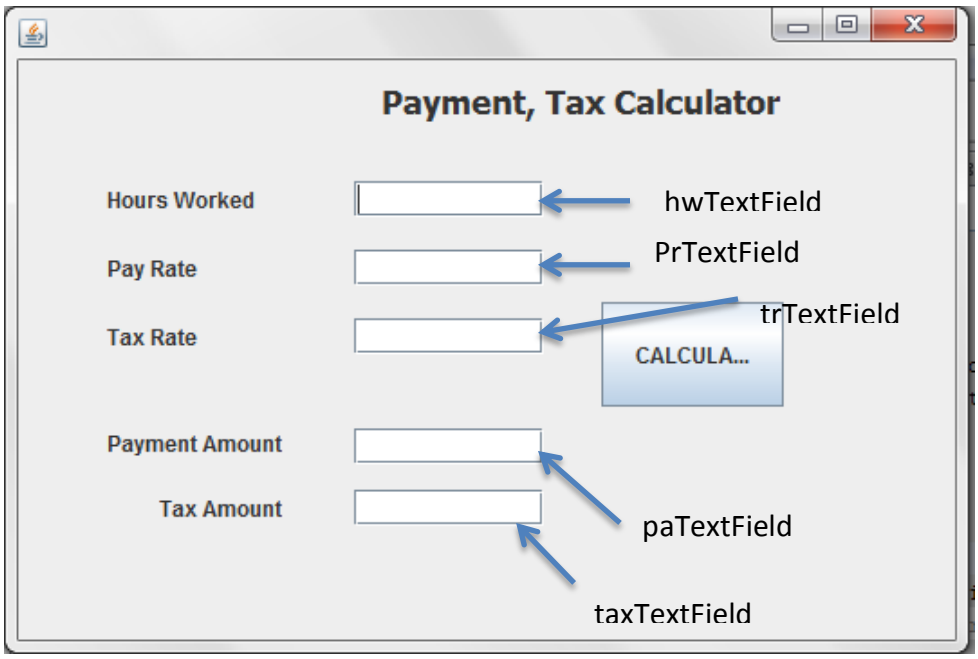
- Suppose we want to change the content of textfield ***classTF*** given above, to "XI", through a code statement; then we use setText( ) method as :

ClassTF.setText("XI");

- The ***setText( )*** changes the value of field before the dot(.) **with the string in its parenthese.**

Example

Design an GUI application that obtains three values in three text fields from user : Hours Worked, Pay Rate and Tax Rate. It should then compute and display Payment Amount (Hours Worked X Pay Rate) and Tax Payable (Payment Amount X Tax Rate) in labels. Assume any numeric data types for these values. The outlook of application should be as shown below :




```
int hours  = Integer.parseInt(hwTextField.getText( )) ;

int prate = Integer.parseInt(prTextField.getText( ));

int trate  = Integer.parseInt (trTextField.getText( ));

int payAmt = hours  * prate ;

int taxAmt  = PayAmt * trate ;

paTextField.setText( " "+payAmt);

txtamtLabel.setText(" "+taxAmt);
```

JOptionPane.showMessageDialog( ) method - Displaying message in a dialog form

- Using this method, we can produce a basic dialog displaying a message to the user. The user will see your message with only an "OK" button to close the dialog.
- **Two steps to use this method** are :

  (a) Firstly, in the source code editor, where you type your code, at top most position type the following line:

  import javax . swing . JOptionPane ;

  (b) Now display desired message as per the following syntax:

  JOptionPane.showMessageDialog (null, "desired-message");

  E.g. to display a message "Hello there!!!", you will write:

  JOptionPane.showMessageDialog (null, "Hello-there!");

CONSTANTS

- Constants are the values that is not going to change when the program is executed. This can be done as :

  final double TAXRATE = 0.25 ;

  The **keyword final makes a variable as constant** i.e., whose value cannot be changed during program execution.

- Once declared constants, their value cannot be modified e.g. after declaring constant TAXRATE, if we issue a statement like :

  TAXRATE = 0.50 ; → Error

  It will cause an error, as the value of constants cannot be modified.

OPERATORS IN JAVA

- The ***operations (specific tasks)*** are represented by **operators** and the ***objects of the operation(s)*** are referred to as **operands.**
- **E.g.** In expression **2 + 3** , operands are **2 & 3** and operator is **+** .

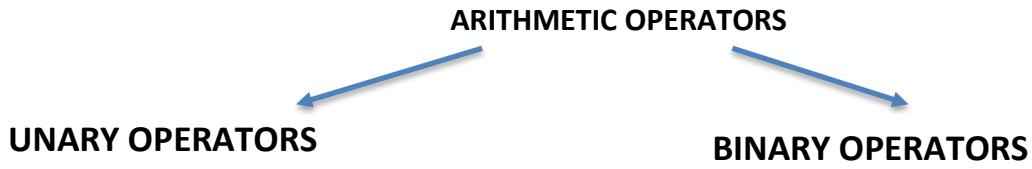
TYPES OF OPERATORS:

1. Arithmetic Operators
   - Unary Operators
   - Binary Operators
2. Increment/Decrement Operators
3. Relational Operators
4. Logical Operators
5. Assignment Operators
6. Conditional Operators

ARITHMETIC OPERATORS

- It provides operators for five basic arithmetic calculations: addition, subtraction, multiplication, division and remainder which are:

+, - , * , / and %.



UNARY

- Operators that act on one operand are referred to as Unary Operators.

1. **Unary +**
   If a = 5 , then +a means 5
   If b = 4, then +b means 4
2. **Unary –**
   If a = 5, then –a means -5
   If a = -4 , then –a means 4

BINARY OPERATORS

- Operators that act upon two operands are referred to as Binary Operators.

1. **Addition operator (+)**
   e.g. 38 + 42
   a + 5 (where a = 2) results in 7.
   a + b (where a=4, b=6) results in 10.
2. **Subtraction operator (-)**
   e.g. 14 – 3 evaluates to 11.
   a – b (where a = 7, b = 5) evaluates to 2.
3. **Multiplication operator (*)**
   e.g. 3 * 4 evaluates to 12.
   b * 4 (where b = 6) evaluates to 24.
4. **Division operator (/)**
   e.g. 100/50 evaluates to 2.
   a / 2 (a=16) evaluates to 8.
   a / b (where a = 15.9, b= 3) evaluates to 5.3
5. **Modulus operator (%)**
   The % operator produces the remainder of dividing the first by the second operand. For example,

   19 % 6 evaluates to 1, similarly, **17 % 5** evaluates to 2.

Operator + with Strings

- When we use the + with numbers, the result is also a number. However, if we use operator + with strings, it concatenates them, e.g.

  5 + 6 results into 11.

"5" + "6" results into "56"

"abc" + "123" results into "abc123"

" " + 5 results into "5"

" " + 5 + "xyz" results into _____.

Increment (++) /Decrement (--) Operators

- The operator **++ adds 1** to its operand, and **-- subtracts one**.
- In other words,

  ++a or **a++** is same as **a = a+1.**

  --a or **a--** is same as **a=a-1**.

- Both increment and decrement operators come in **two varieties**:
  1. **Prefix version :** In this, operator comes before the operand, as in ++a or --a.
  2. **Postfix version :** In this, operator comes after the operand, as in a++ or a --.

The two versions have the same effect upon the operand, but they differ when they take place in an expression.

Working with prefix version (*Principle: Change then use*)

- When an increment or decrement operator precedes its operands, Java performs the increment or decrement operation before using the value of operand.
- E.g. in expression **sum = sum + (++count) ;**
  (assuming initial value of sum and count is 0 and 10)

sum = sum + (++count)
sum = 0 + 11
sum = 11

Value of count is incremented by 1, then used.

Working with postfix version (*Principle: use then Change*)

- When an increment or decrement operator follows its operand, Java first uses the value of the operand in evaluating the expression before incrementing or decrementing the operand's value.
- E.g. in expression **sum = sum + (count++);**
  (assuming initial value of sum and count is 0 and 10)

sum = sum + (count++)
sum = 0 + 10
sum = 10

Value of count is used here, then it will be incremented later.

Q1. Evaluate the expression **x = ++y + 2*y** , if **y = 6**.

Q2. What will be the value of **P=P* ++J** where **J is 22** and **P=3** initially ?

Q3. What will be the value of following, if j =5 initially ?

(i) (5 * ++j) % 6 (ii) (5 * j++) % 6

Q4. What will be the value of **j= --k + 2 * k ,** if **k is 20** initially ?

Q5. Will the value of Y be the same for the two cases given below ?

(i) Y = ++X (ii) Y = X ++ (Given the value of X is 42).

Relational Operators

- Relational operators determine the relation among different operands. Java provides **six relational operators** for comparing numbers and characters.

  < (less than)
  <= (less than or equal to)
  == (equal to)
  >(greater than)
  >= (greater than or equal to)
  != (not equal to)

- If the comparison is true, the relational expression results into the Boolean value ***true*** and to boolean value ***false***, if the comparison is false. **E.g**.

  4 > 3 (true)
  3 > 4 (false)
  7==0 (false)
  3==3 (true)

- **Relational operators don't work with Strings. E.g.**
  "123" > "90"
  "5" < "3"

Difference between = & == operator

== is equal to operator and is used for comparison. **e.g.**

value == 3, this expression tests whether the value is equal to 3 ?
The expression produce result true if the comparison is true and boolean false if it is false.

= is assignment operator, and is used to assign value to a variable. E.g.

value = 3, this expression assigns 3 to variable value.

Logical Operators

- Relational operators often are used with logical operators to construct more complex expressions. Three Logical Operators are :
  1. || **(OR Operator)**
  2. && **(AND Operator)**
  3. ! **(NOT Operator)**

The logical OR operator (||)

- The logical OR operator (||) combines two expressions which make its operand. The OR operator evaluates to boolean true if either of its operand evaluates to true. **E.g.**

(4 == 4) || (5 == 8) results into true because first expression is true.

(1 == 0) || (0 > 1) results into false because neither expression is true.

The logical AND operator (&&)

- The logical AND operator (&&) also combines two expressions. The AND operator evaluates to boolean true only if both operands evaluates to true. **E.g.**

(6 == 3) && (4 == 4) results into false because first expression is false.

(6 < 9) && (4 > 2) results into true because both expressions are true.

The logical NOT operator (!)

- The logical NOT operator, written as **!** works on single expression or operand , i.e. it is a unary operator.
- It negates or reverses the truth value of the expression following it. e.g.
  ! (5 > 2) results into false because expression 5 > 2 is true.
  !(5>9) results into true because the expression 5 > 9 is false.

Assignment Operator

- The assignment operator , = is used to assign one value to another, e.g.

```
int x , y , z ;
x =9 ;
y = 7;
z = x + y ;
```

Java Shorthand Operators

- Java male use of shorthand operators to simplify certain assignment operators. E.g.

a = a + 10 ;
can be written as
a += 10 ;
The operator += tells the compiler to assign to a the value of a+10. This shorthand *works for all binary operators* in Java.

e.g.

x -= 10 ; is equivalent to x = x – 10 ;
x *= 3 ; is equivalent to x = x * 3 ;
x /= 2 ; is equivalent to x = x/2 ;
x %= z ; is equivalent to x = x % z ;

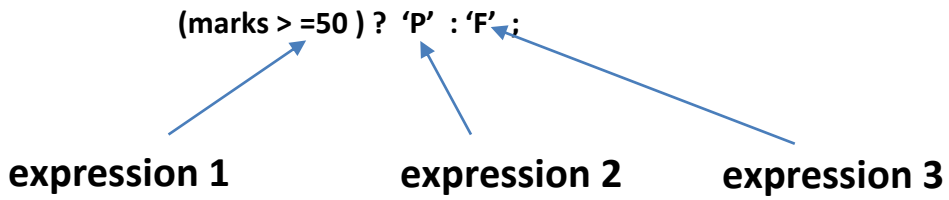
CONDITIONAL OPERATOR (? :)

- This operator store a value depending upon a condition. This operator is ternary operator i.e. it requires three operands.
- The general form of conditional operator is :

expression 1 ? expression 2 : expression 3 ;

If expression1 evaluates to true, i.e., then the value of whole expression is the value of expression2, otherwise the value of the whole expression is the value of expression3.

e.g.



The identifier ***result*** will have the value 'P' if the expression1 ***marks>=50*** evaluates to true, otherwise ***result*** will have the value 'F'.

Other examples:

(6 > 4) ? 9 : 7 ;

(4 == 9) ? 10 : 25 ;

- The conditional operator can be nested also, i.e. any of the expression2 or expression3 can itself be another conditional operator.

e.g.

(class >= 10) ? (marks >= 80 ? 'A' : 'B') : 'C' ;

The expression first tests the condition if class >=10, if it is true, it tests for another condition if marks >= 80.

Operator Precedence

- It determines the order in which expressions are evaluated. E.g. in following expression :

  y = 6 + 4/2 ;

  the division operation takes place first, rather than addition.

Operator Associativity

- Associativity rules determine the grouping of operands and operators in an expression with more than one operator of the same precedence.
- ***For most operators , the evaluation is done left to right,*** i.e.

  x = a + b – c ;

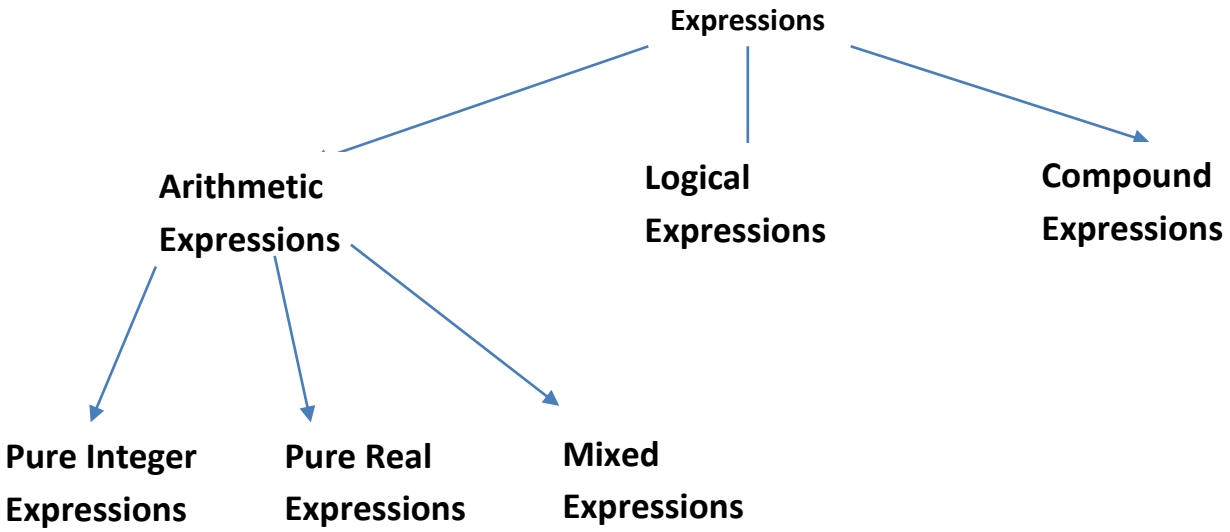
  Here addition and subtraction have the same precedence rating and so a and b are added and then from this sum c is subtracted.

Descending Order of Operator Precedence

Operator	Notes	Associativity
* / %	Multiplication,Division, Modulus	Left to Right
+ -	Addition , Subtraction	Left to Right
< > >= <=	Relational Comparison tests	Left to Right
== !=	Equality	Left to Right
&&	AND Operator	Left to Right
||	OR Operator	Left to Right
? :	Conditional Operator	Right to Left
=	Assignment Operator	Right to Left

Expressions

- An expression in java is any valid combination of operators, constants and variables, i.e. a legal combination of Java tokens.



Arithmetic Expressions

- It may have two or more variables or constants, or two or more expressions joined by valid arithmetic operators. E.g.

  b * c

  (b+9) * (c-d)

  (a+b+c)/d

Pure Integer Expression

- In pure Integer Expression, all operands are of integer type. E.g.

  int a=6, b=8, c=9 ;

  a + b ;

  a + b + c ;

  a * b ;

Pure Real Expression

- In pure Real Expression, all operands are of real(float/double) type. E.g.

  float p=2.4, q=3.6, r=7.2 ;

  p + q ;

  p + q + r ;

  p * q ;

** **pure expressions produce the result having the same datatype as that of its operand. e.g.**

int a=5, b=2 , c ;

a + b will produce result 7 of int type.

a / b will produce result 2 of int type.

Mixed Expression

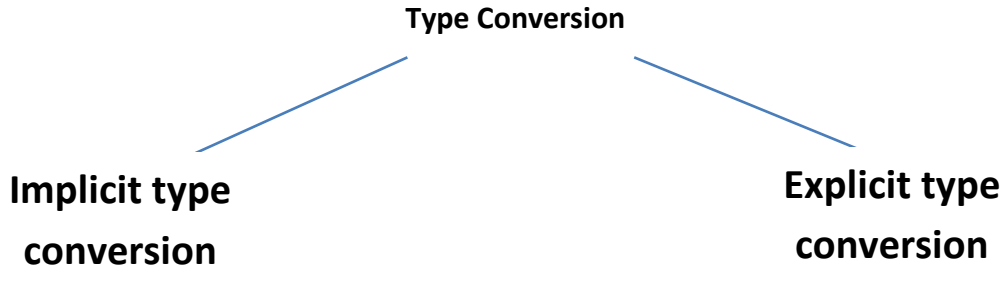
- In mixed expression, the operands are of mixed or different data types. E.g.

```
int z = 9 ;
float c = 5.4 ;
z  + c ;
```

Mixed Expression

Type Conversion

- When constants and variables of different types are mixed in an expression, **they are converted to the same type.**
- The process of converting one predefined type into another is called **Type Conversion.**



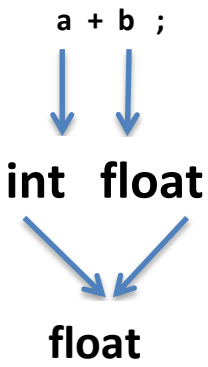
Implicit type conversion

- It is a conversion **performed by the complier without programmer's intervention.**
- This is done as per the **following type conversion rules** :

- If either operand is of type double, the other is converted to double.
- Otherwise, if either operand is of type float, the other is converted to float.
- Otherwise, if either operand is of type long, the other is converted to long.
- Otherwise both operands are converted to type int.

e.g. int a =3 ;

float b = 3.5 ;



It means that **data type of result** in above expression will be **float.**

Q: Given the following set of identifiers:

byte b ;
char ch ;
short sh ;
int intval ;
long longval ;
float fl ;

Identify the datatype of the following expressions:

(a) 'a' – 3
(b) intval * longval – ch ;
(c) fl + longval/sh ;

Explicit Conversion

- This conversion is user defined that **forces an expression to be of specific type.**
- Explicit conversion is done as shown below :

(type) expression

where ***type*** is a valid data type to which conversion is to be done.

e.g. int a= 2 ; int b = 4.5 ;

a + b ; produces result as 6.5, but in order to obtain result as 6 , we explicitly convert result into int.

To make sure that the expression a + b evaluates to type int, we write it as :

(int) a+ b ;

Logical Expressions (Boolean expressions)

- The expressions that result into false or true are called Boolean expressions or relational expressions. **E.g.**

  (i) x > y
  (ii) (x > y) && (y < z)
  (iii) (x == 3)

Compound Expression

- A compound expression is the one which is made up by combining two or more simple expressions with the help of operator. **E.g.**
  - (a + b) / (c + d)
  - (a > b) || (b > c)

Maths functions available in Java

Functions	Description
pow(x , y)	This function returns x raised to y. (x^y)
exp(x)	This function returns e raised to x (e^x)
log(x)	This function returns the logarithm of x.
sqrt(x)	This function returns the square root of x.
abs(a)	This function returns the absolute value of a.

Q: Write the corresponding Java expression for the following mathematical expressions:

(i) $\sqrt{a^2+b^2+c^2}$

(ii) $2- ye^{2y} + 4y$

(iii) $p + q/(r+s)^4$

(iv) $|e^x - x|$

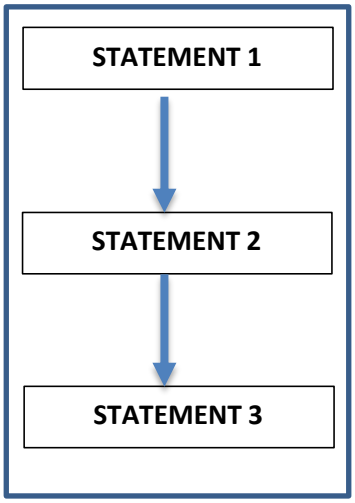
FLOW OF CONTROL

PROGRAMMING CONSTRUCTS

- In a program, statements may be executed sequentially, selectively or iteratively.
- Every programming language provides constructs to support sequence, selection or iteration.

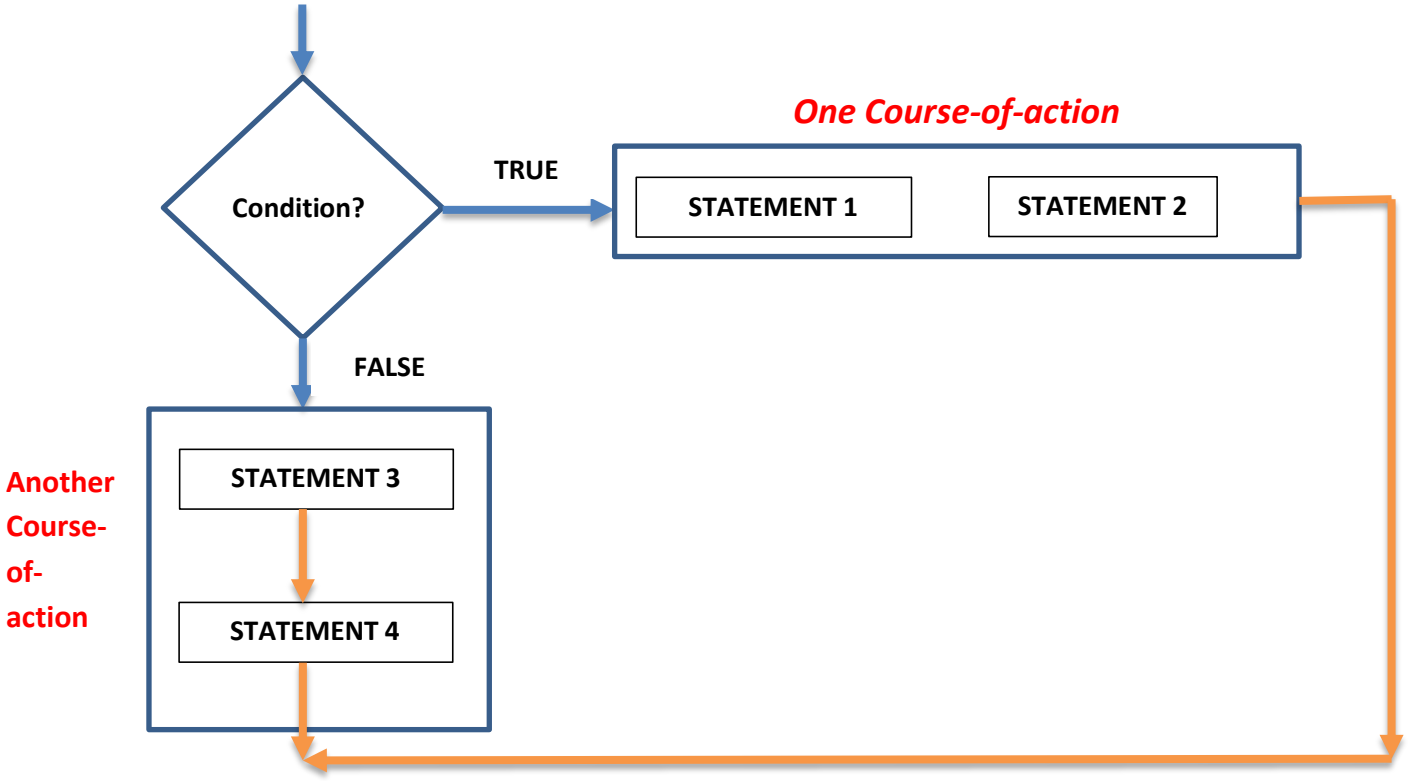
SEQUENCE

- The sequence construct means the statements are being executed sequentially.
- Java execution starts with first statement, and then each statement is executed in turn.



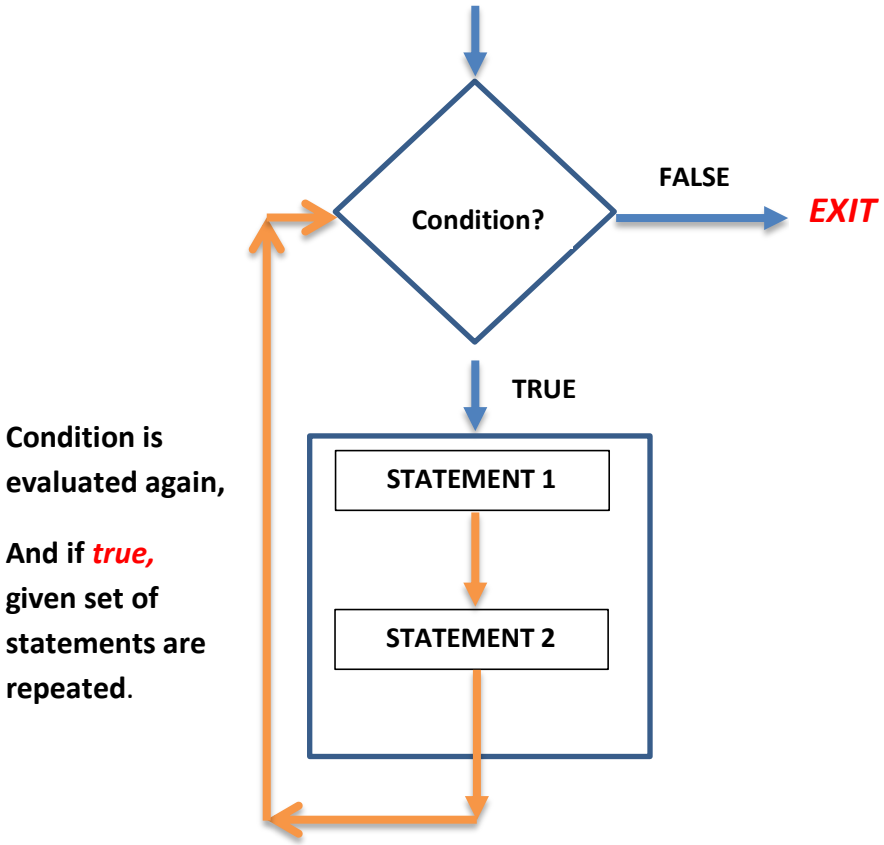
SELECTION

- The selection construct means the execution of statement(s) depending upon a condition-test.
- If a condition evaluates to true, a course of action (a set of statements) is followed otherwise another course of action (a different set of statements) is followed.



ITERATION

- The iteration construct means **repetition** of a set of statements depending upon a condition-test. Till the time a condition is true, a set of statements are repeated again and again. As soon as the condition becomes false, the repetition stops.
- The iteration construct is also called **looping construct**.



- The set of statements that are repeated again and again is called the **body of loop**.

SELECTION STATEMENTS

- The selection statements allow to choose the set of instructions for execution depending upon an expression's truth value.
- Java provide two types of selection statements : **if** and **switch**.

The if statement of Java

- An if statement tests a particular condition; if the condition evaluated to true, a statement or set of statements is executed.
- The Syntax (general form) of the if statement is as shown below :

```
if (boolean expression/condition)
        statement ;
```

where a statement may consist of a single statement or a compound statement. The condition must be enclosed in parentheses. If the condition evaluates to true, the statement is executed, otherwise ignored.

e.g.
```
if (ch == 2)
    R.setText("It is number 2");
```

e.g.2 Write code to test whether a number stored in variable **Num** is positive or not. Make use of if statement only.

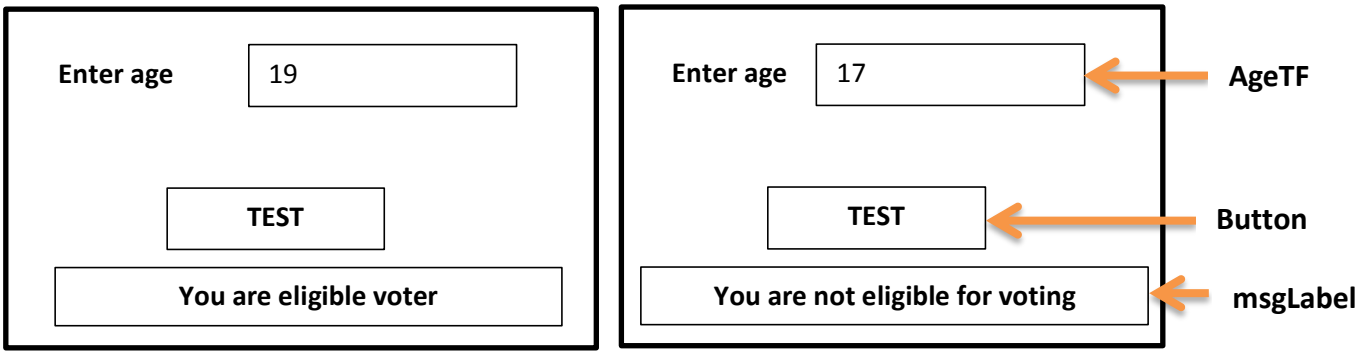
```
if (Num > 0)
  RTF.setText ("Number is positive");
```

Explanation

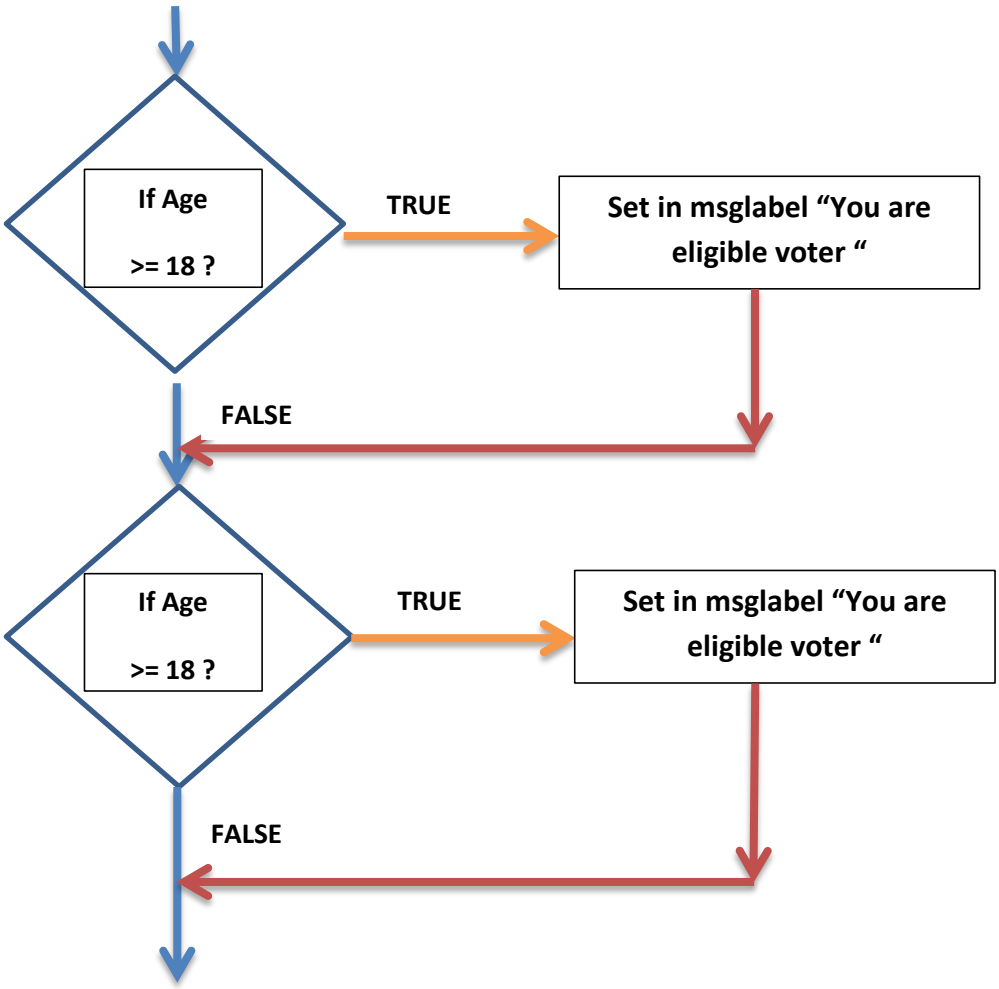
The if statement makes a decision , if the value of the logical expression Num > 0 is ***true***, the statement (***RTF.setText ("Number is positive");)*** gets executed ; if the value of the logical expression is ***false,*** it does not execute the statement. That means no message is printed when the condition (***Num > 0***) is false.

Question:

Obtain age of a person and then display whether he/she is eligible for voting.

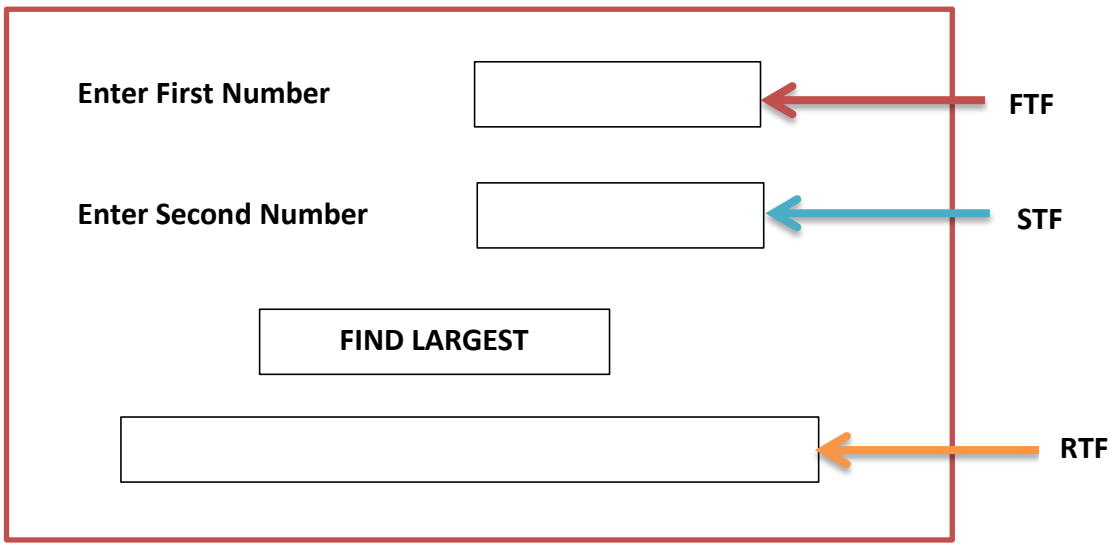



```
int age ;
  String txt = ageTF.getText ( );
  age = Integer.parseInt(txt);
    if (age >= 18)
       msgLabel.setText ("You are eligible voter") ;
    if (age < 18)
       msgLabel.setText("You are not eligible for voting");
```



Question:

Find the largest of two numbers.

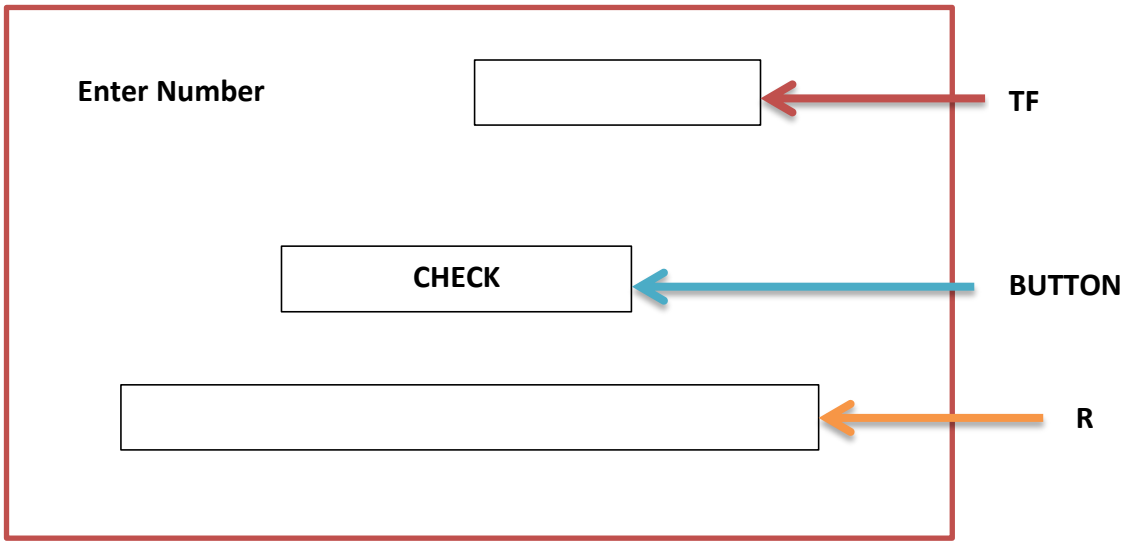


Solution:

```
String a = FTF.getText ( );
 int n1 = Integer.parseInt (a);
String b = STF.getText( );
 int n2 = Integer.parseInt (b) ;
    if (n1 > n2)
        RTF.setText(" "+n1) ;
    if (n2 > n1)
        RTF.setText(" "+n2) ;
```

Question:

Find whether number entered in textfield is positive,negative or zero.



Solution:

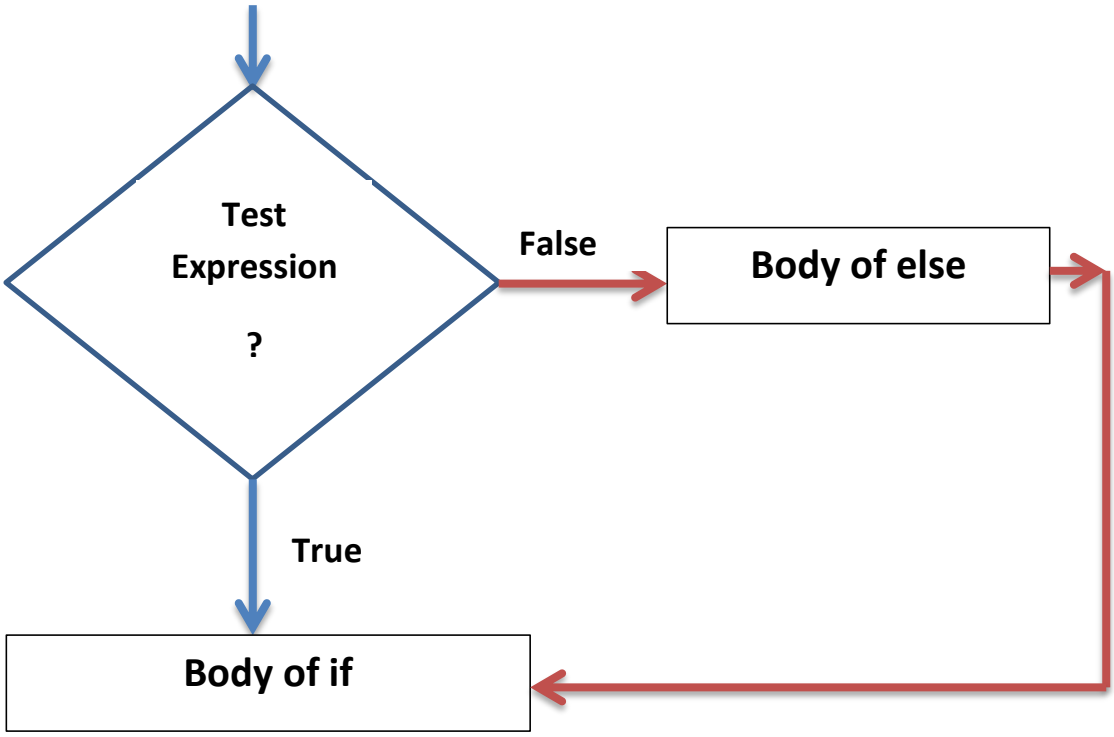
```
String q = TF.getText ( );
 int n = Integer.parseInt (q);
      if (n > 0)
          R.setText ("Number is positive");
      if (n < 0)
           R.setText ("Number is negative");
       if (n == 0)

           R.setText ("Number is zero");
```

if else statement

- The syntax (general form) of if else statement is the following :

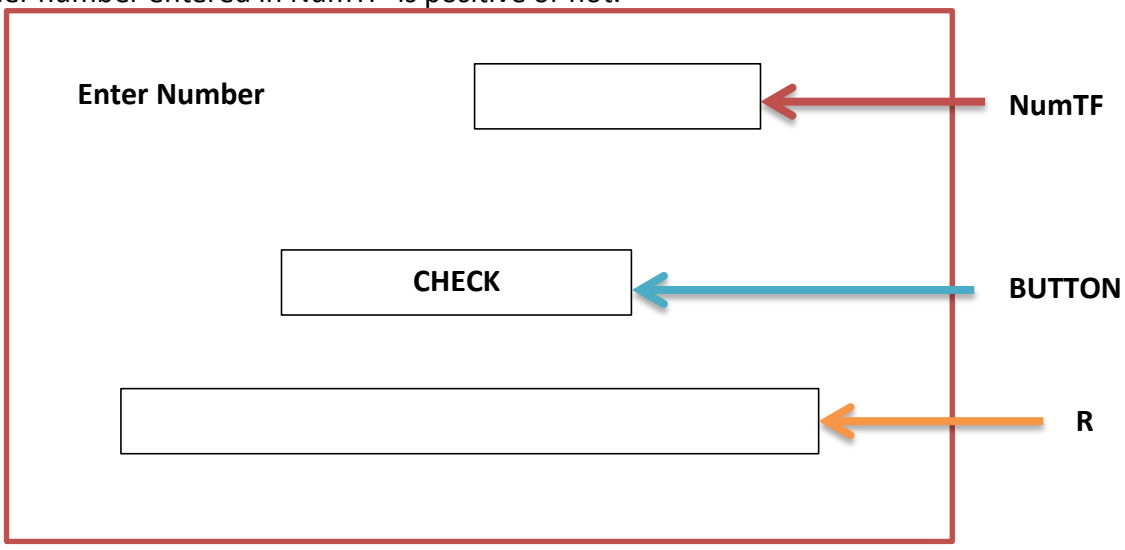
```
if (expression)
    statement 1 ;
else
    statement 2 ;
```

If the **expression** evaluates to ***true***, the statement 1 is executed, otherwise statement 2 is executed. The statement 1 and statement 2 can be single statement or compound statement.



Question:

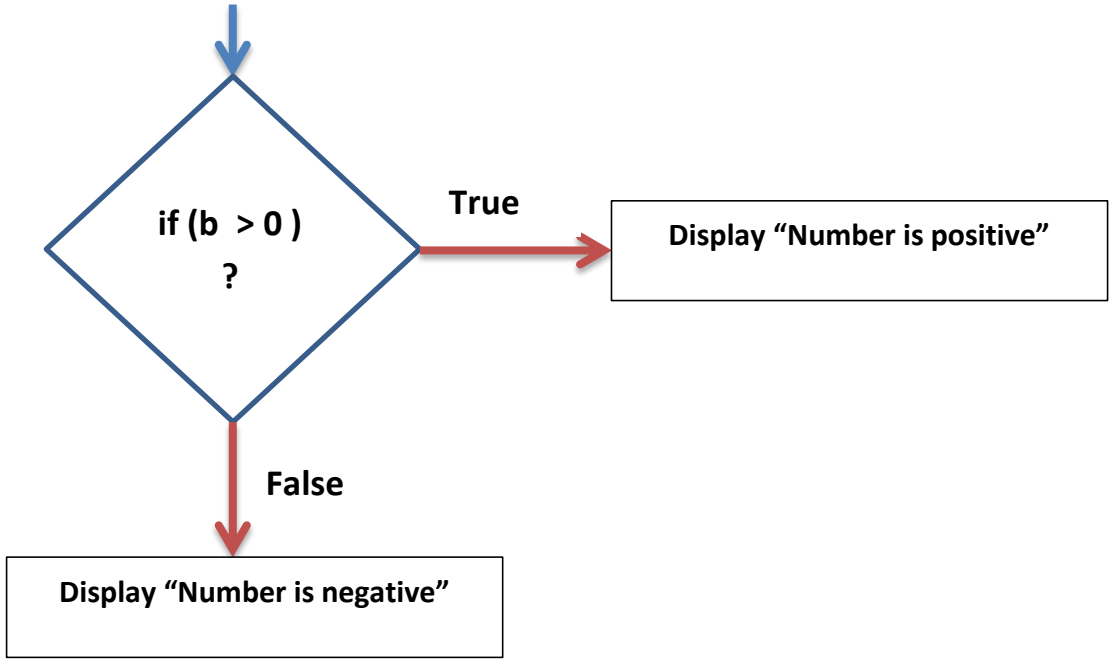
Find whether number entered in NumTF is positive or not.



Solution:

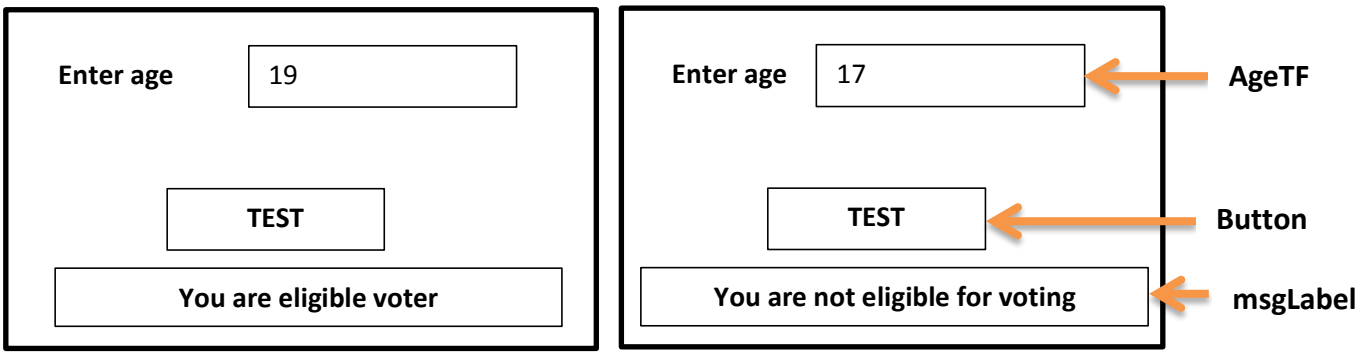
```
String a = NumTF.getText ( );
int b = Integer.parseInt(a);
if (b > 0 )
        R.setText( " Number is positive ");
else
        R.setText( " Number is negative ");
```

Explanation



Question:

Obtain age of a person and then display whether he/she is eligible for voting.**(using if else)**

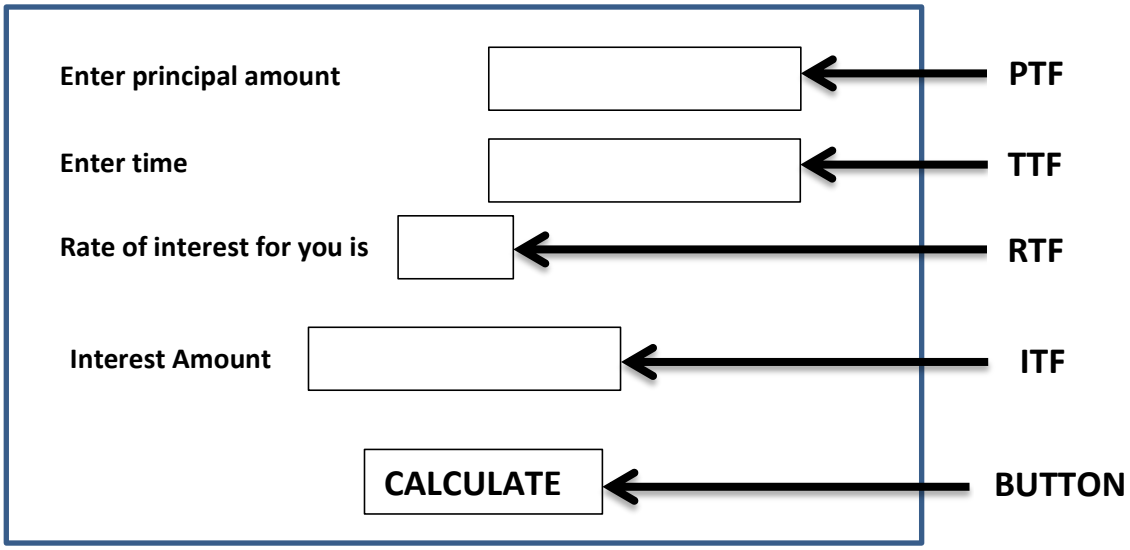


Solution:

```
String x = AgeTF.getText ( );
int age = Integer.parseInt(x);
  if (age >= 18)
        msgLabel.setText("You are eligible voter");
 else
        msgLabel.setText("You are not eligible for voting");
```

Question:

Obtain principal amount and time and calculate simple interest as per following specifications: **If principal is greater than or more than Rs.10000**, then rate of interest is 6% otherwise it is 5%.



Solution:

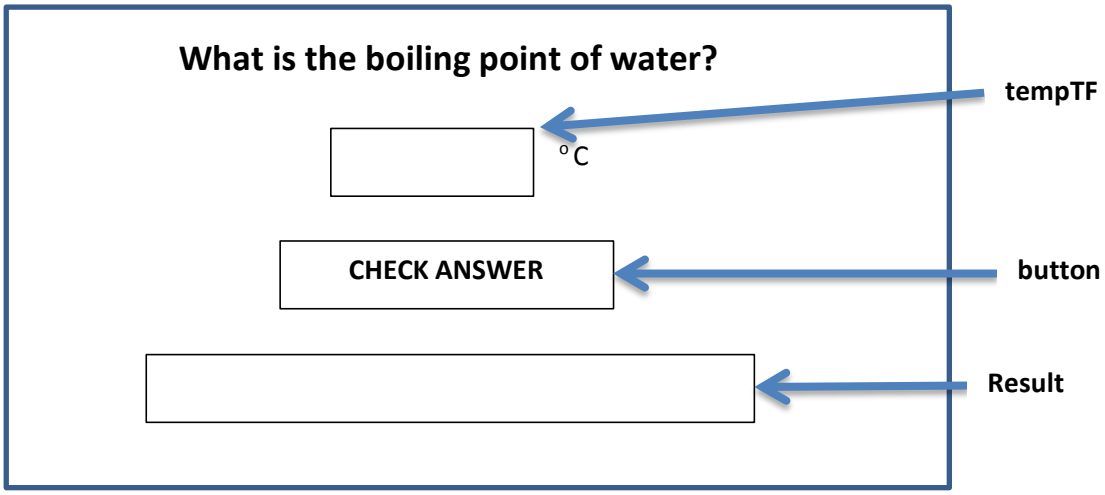
```
double rate;
String t =  PTF.getText( );
int prin = Integer.parseInt(t);
String g = TTF.getText( );
int time = Integer.parseInt(g);
      if (prin >= 12000)
      {
          RTF.setText("6%");
          rate =0.06 ;
       }
      else
       {
           RTF.setText("5%");
           rate =0.05 ;
       }
        double intr = prin * time * rate ;
        ITF.setText(" " + intr);
```

***** the curly brackets are not required if only ONE statement follows if or else, but in case of multiple statements, curly brackets are required.***

Question:

Program to obtain boiling point of water from user and report whether the user has entered the correct answer or not. Make use of if else statement.

(Hint : Boiling point of water is 100^{o} C).



Solution:

```
String t = tempTF.getText( );
int temp = Integer.parseInt(t);

  if(temp == 100)
        Result.setText("You are right. It is 100° C") ;
  else
        Result.setText("Your answer is wrong") ;
```

Nested if

- A nested if is an if that has another if in its if's body or in its else's body.
- The nested if can have one of the following three forms:

1. **If statement nested within if body**

```
    if(expression1) {

          if(expression2)
                statement 1 ;
          [ else
                statement 2 ; ]
    }
      else
            body of else ;
```

2. If statement nested within else body

```
if(expression 1)
        body of if ;

      else {
            if(expression2)
              statement 1;
           [ else
              Statement 2; ]
         }
```

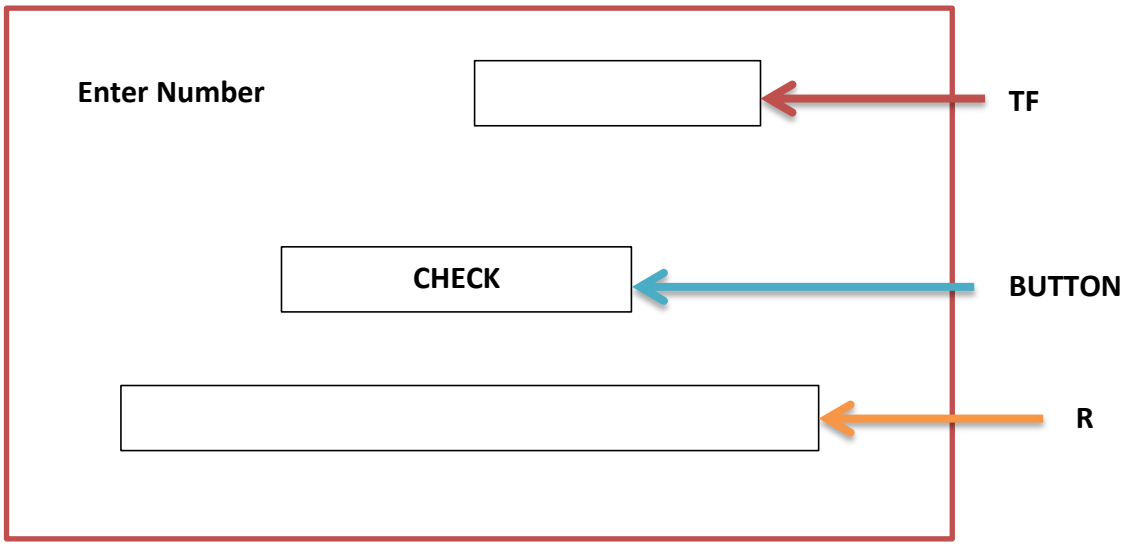
3. If statement nested within if and else body

```
if (expression1) {

        if(expression2)
          statement 1;
        [ else
            statement 2 ; ]
    }
      else {

        if(expression2)
          statement 1;
        [ else
            statement 2 ; ]
  }
```

** **The part in [ ] means, it is optional.**

Question:

Find whether number entered in textfield is positive,negative or zero. (using nested if)

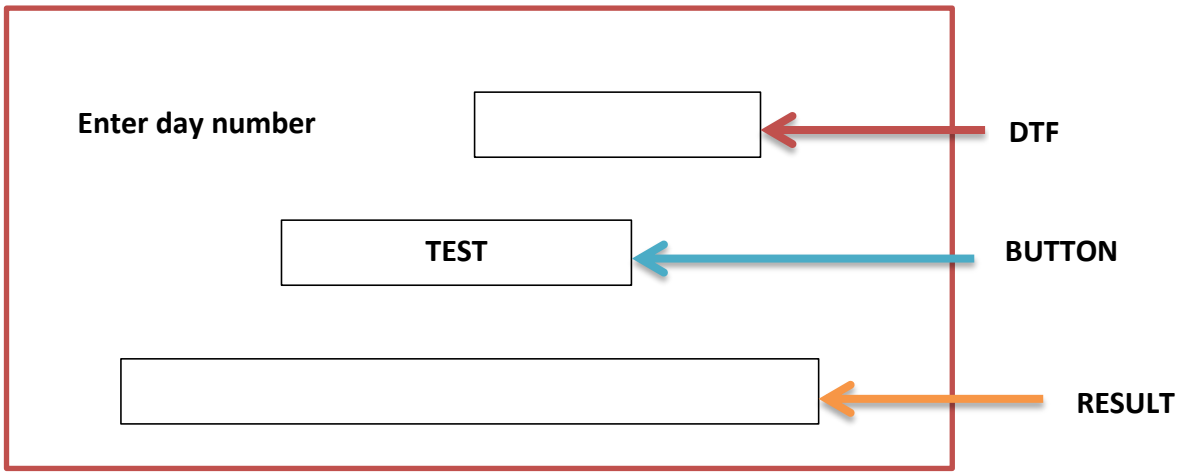


Solution:

```
String q = TF.getText ( );
int n = Integer.parseInt (q);
    if (n > 0)
       R.setText (“Number is positive”);
    else {
          if(n < 0)
             R.setText (“Number is negative”);
          else
             R.setText (“Number is zero”);
          }
```

Question:

Write code to read the Day number and display the weekday. E.g. for 1 -> Sunday, 2-> Monday etc.



Solution:

Using if statement only

```
  String a = DTF.getText( );
int  w = Integer.parseInt(a);
if(w == 1)
      RESULT.setText(“Day is Sunday”);
 if(w == 2)
      RESULT.setText(“Day is Monday”);
 if(w == 3)
      RESULT.setText(“Day is Tuesday”);
 if(w == 4)
      RESULT.setText(“Day is Wednesday”);
 if(w == 5)
      RESULT.setText(“Day is Thursday”);
 if(w == 6)
      RESULT.setText(“Day is Friday”);
  if(w == 7)
      RESULT.setText(“Day is Saturday”);
```

Using nested if statement

```
String a = DTF.getText( );
int  w = Integer.parseInt(a);
if(w == 1)
      RESULT.setText("Day is Sunday");
else {
       if(w == 2) {
          RESULT.setText("Day is Monday");
                }
         else {
                    if(w == 3) {
                 RESULT.setText("Day is Tuesday");
                      }
              else {
                    if(w == 4) {
                     RESULT.setText("Day is Wednesday");
                         }
                 else {
                    if(w == 5) {
                     RESULT.setText("Day is Thursday");
                         }
                          else {
                        if(w == 6) {
                           RESULT.setText("Day is Friday");
                                }
                             else {
                            if(w == 7) {
                               RESULT.setText("Day is Saturday");
                                  }
                          }
                      }
                   }
               }
          }
```

The if else if ladder

- It takes the following general form :

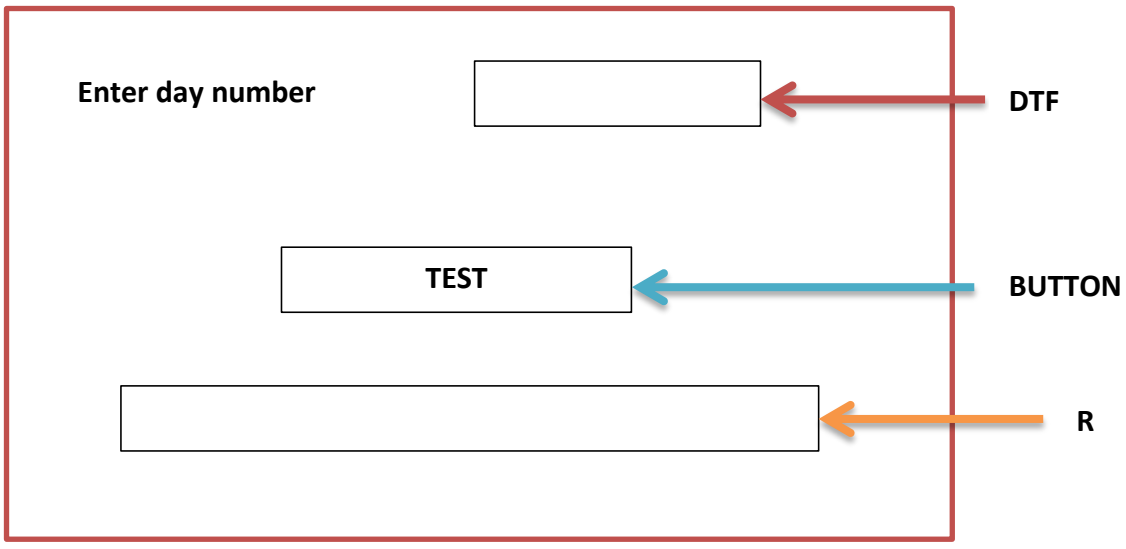
```
if (expression 1)
      statement 1;
else if (expression 2)
      statement 2;
 else if (expression 3)
      statement 3;
     .
     .
     .
else
      statement 4 ;
```

The ***expressions*** are evaluated from the top downward. As soon as an ***expression*** evaluates to **true**, the statement associated with it is executed and the rest of the ladder is bypassed or ignored. If **none of the expressions are true**, the ***final else*** gets executed.

e.g.

Question:

Write code to read the Day number and display the weekday. E.g. for 1 -> Sunday, 2-> Monday etc. **(Using if else if ladder)**



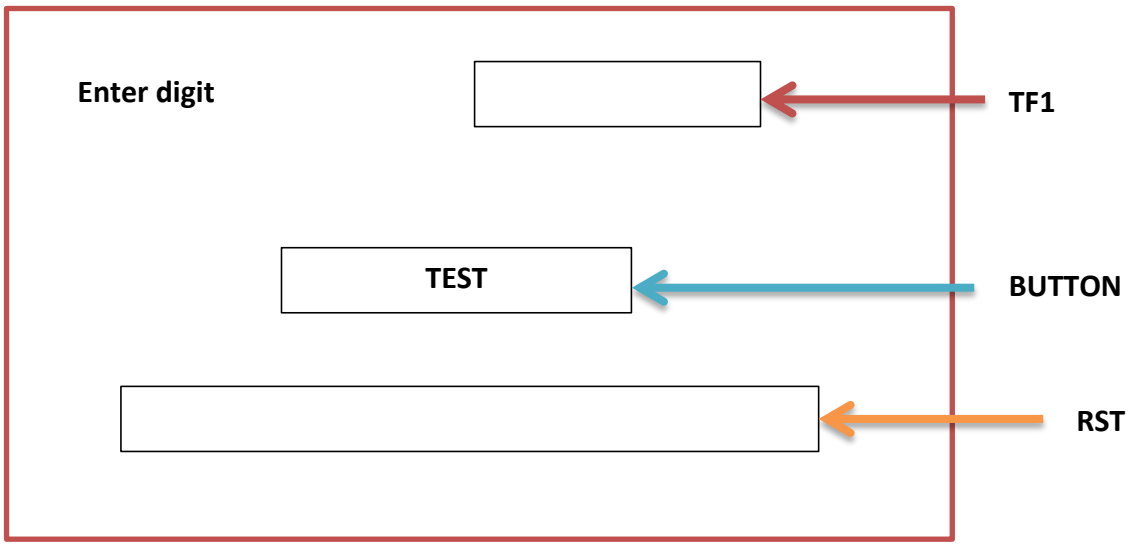
Solution:

```
String a = DTF.getText( );
int  w = Integer.parseInt(a);
if(w == 1)
      RESULT.setText("Day is Sunday");
```

```
else if(w == 2)
      RESULT.setText("Day is Monday");
else if(w == 3)
      RESULT.setText("Day is Tuesday");
else if(w == 4)
      RESULT.setText("Day is Wednesday");
   else if(w == 5)
     RESULT.setText("Day is Thursday");
   else if(w == 6)
      RESULT.setText("Day is Friday");
   else if(w == 7)
      RESULT.setText("Day is Saturday");
   else
      RESULT.setText("Please Enter no. between 1 & 7");
```

Question:

Obtain a single digit (0 – 9) and display it in words. **(Using if else if ladder)**



Solution:

```
String z = DTF.getText( );
int  dig = Integer.parseInt(z);
if(dig == 0)
      RESULT.setText("Digit is zero");
 else if(dig == 1)
      RESULT.setText("Digit is one");
 else if(dig == 2)
      RESULT.setText("Digit is two");
 else if(dig == 3)
      RESULT.setText("Digit is three");
   else if(dig == 4)
```

```
    RESULT.setText("Digit is four");
else if(dig == 5)
    RESULT.setText("Digit is five");
else if(dig == 6)
    RESULT.setText("Digit is six");
else if(dig == 7)
    RESULT.setText("Digit is seven");
else if(dig == 8)
    RESULT.setText("Digit is eight");
else if(dig == 9)
    RESULT.setText("Digit is nine");
else
    RESULT.setText("Please enter digit between 0 and 9");
```

The switch statement

- It is a multiple branch selection statement.
- This selection statement successively tests the value of an ***expression*** against a list of ***integer or character constants***. When a match is found, the statements associated with that constant are executed.

- **Syntax of switch statement is as follows:**

```
switch (expression)
{
   case constant1 :   statement1;
                      break ;
   case constant2 :   statement2 ;
                      break;
   case constant3  :   statement2 ;
                      break ;
    .
    .
    .
    default : statement ;
}
```

- The ***expression*** is evaluated and its values are matched against the values of the ***constants specified in the case statements***. When a match is found, the statement associated with that case is executed until the break statement or the end of switch statement is reached.

- The **default statement gets executed when no match is found**.

- If the program control flows to the next case below the matching case, in the absence of break, this is called **fall through.**

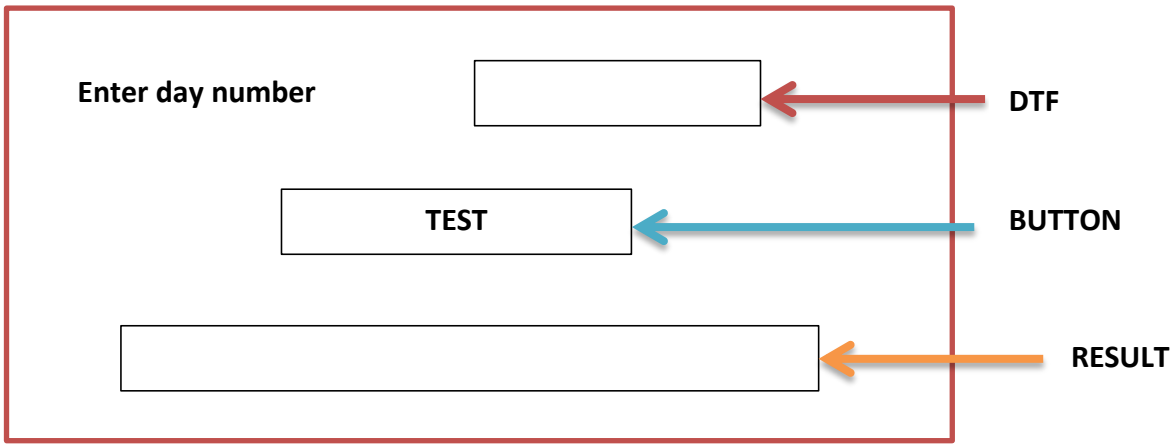
Points to remember:

1. A **case statement** cannot exist by itself, **outside a switch**.
2. The **break statement**, used under switch, is one of **Java Jump Statements**.
3. ***When a break statement is encountered in a switch statement, program execution jumps to the line of code outside the body of switch.***

e.g.1.

Question:

Write code to read the Day number and display the weekday. E.g. for 1 -> Sunday, 2-> Monday etc. **(Using switch statement)**

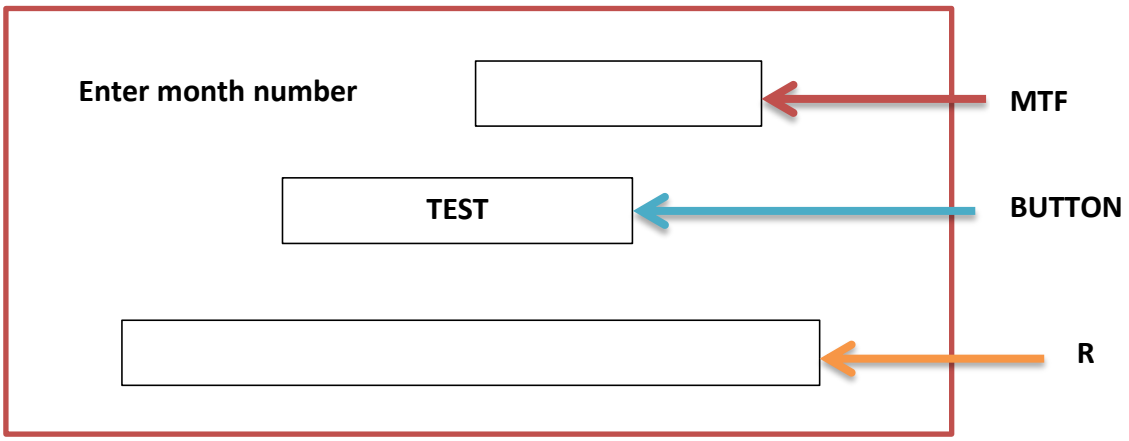



```
String a = DTF.getText( );
int  w = Integer.parseInt(a);
 switch(w) {

        case 1 : RESULT.setText("Day is Sunday");
                 break ;
        case 2 : RESULT.setText("Day is Monday");
                 break ;
        case 3 : RESULT.setText("Day is Tuesday");
                 break ;
        case 4 : RESULT.setText("Day is Wednesday");
                 break ;
        case 5 : RESULT.setText("Day is Thursday");
                 break ;
        case 6 : RESULT.setText("Day is Friday");
                 break ;
        case 7 : RESULT.setText("Day is Saturday");
                 break ;
    default : RESULT.setText("Enter Number between 1&7");

}
```

e.g.1.

Question:

Write code to read the months number and display the name of month . E.g. for 1 -> January, 2-> February, 3-> March, etc.**(using switch statement)**




```
String x = MTF.getText( );
int  m = Integer.parseInt(x);
switch(w) {
        case 1 : R.setText("Month is January");
                 break ;
        case 2 : R.setText("Month is February");
                 break ;
        case 3 : R.setText("Month is March");
                 break ;
        case 4 : R.setText("Month is April");
                 break ;
        case 5 : R.setText("Month is May");
                 break ;
        case 6 : R.setText("Month is June");
                 break ;
        case 7 : R.setText("Month is July");
                 break ;
        case 8 : R.setText("Month is August");
                 break ;
        case 9 : R.setText("Month is September");
                 break ;
        case 10 : R.setText("Month is October");
                 break ;
         case 11 : R.setText("Month is November");
                   break ;
         case 12 : R.setText("Month is December");
                   break ;
      default : R.setText("Enter Number between 1&12");

}
```

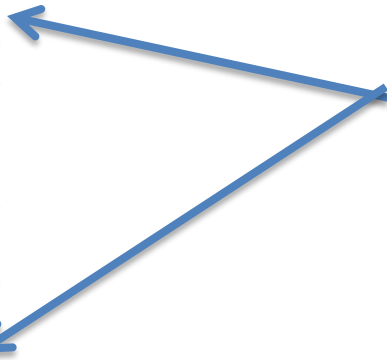
Difference between switch and if else

	switch statement	if else statement
1.	switch statement can only test for Equality.	if else statement can evaluate a relational or logical expression.
2.	switch statement cannot handle floating point tests.The case labels of switch must be byte, short, int or char.	if else statement can handle floating tests apart from integer test.
3.	switch case label value must be a constant.	if two or more variables are to be compared, use if else statement.

Some important points about switch statement

- Multiple identical case expressions are not allowed. E.g.

```
switch(val) {
        case 5 :
              .
              .
        case 6 :
              .
              .
        case 7 :
              .

        case 5 :
              .
              .
         }
```



Two identical case expressions are not allowed

Example to illustrate the working of switch in the absence of break statement.

What will be the output of following code fragment if the value of ch is
(i) a (ii) c (iii) d (iv) h (v) b

```
switch(ch)  {
            case 'a' : System.out.println("It is a");
            case 'b' : System.out.println("It is b");
            case 'c' : System.out.println("It is c");
                       break ;
            case 'd' : System.out.println("It is d");
                       break ;
            default : System.out.println("Not abcd");

          }
```

Conversion from if else to switch & switch to if else

Rewrite the following code fragment using switch:

```
if(a == 0)
   System.out.println("Zero");
if(a == 1)
   System.out.println("One");
if(a == 2)
   System.out.println("Two");
if(a == 3)
   System.out.println("Three");
```

Rewrite the following code fragment using if :

```
String Remarks;
    int code = Integer.parseInt(jTextField1.getText( ) );
    switch (code)
     {
      case 0 : Remarks ="100% Tax Exemption";
                break;
      case 1 : Remarks = "50% Tax Exemption";
                break;
      case 2 : Remarks ="25% Tax Exemption";
               break;
      default : Remarks = "Invalid Entry!";
     }
```

ITERATION STATEMENTS

- The iteration statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. The iteration statements are also called loops or looping statements.
- Java provides **three kinds of loops :** **1.** for loop **2.** while loop **3**. do while loop

ELEMENTS THAT CONTROL A LOOP (PART OF A LOOP)

1. Initialization Expression

- The initialization expression give the loop variable their first value.

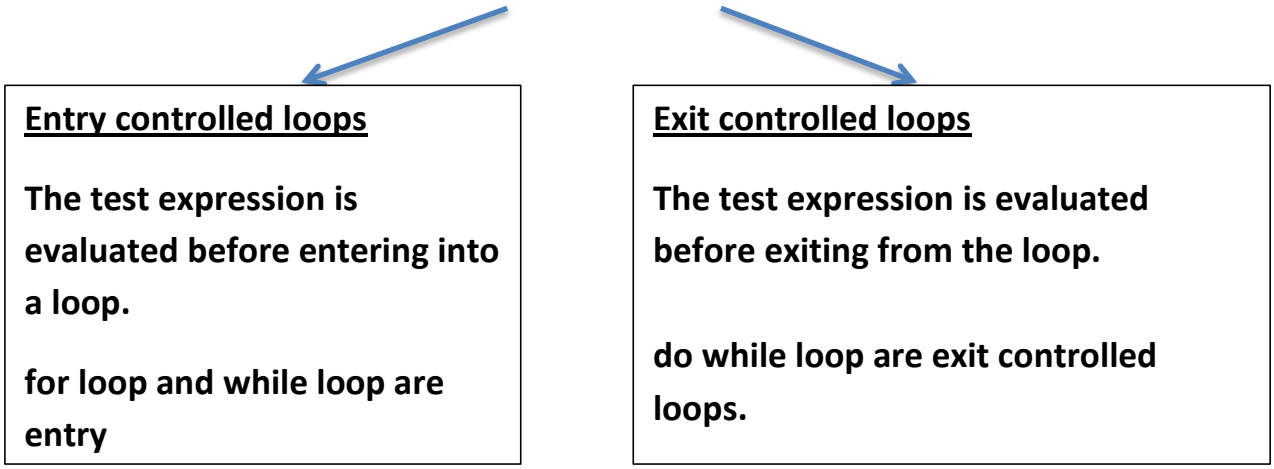
- The initialization expression is executed only once, in the beginning of the loop.

2. Test Expression

- The test expression is an expression whose truth value decides whether the loop body will be executed or not.

- If the test expression evaluates to true the loop body gets executed, otherwise the loop is terminated.

TYPES OF LOOPS



3. Update expression

- The update expression change the value of loop variable.

- The update expression is executed at the end of the loop after the loop body is executed.

4. The Body of the Loop

- The statements that are executed repeatedly form the body of the loop.

for loop

The Syntax of the for loop is :

for(**initialization expression ; test expression ; update expression**)

body of loop ;

e.g.1. Write code using for loop to print numbers from 1 to 10.

Initialisation expression

Test expression

Update expression

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

Upon execution, the above code will yield :

1 2 3 4 5 6 7 8 9 10

Explanation

1. First , initialization expression is executed i.e. i = 1 which gives the first value 1 to variable i.

2. Then, the test expression is evaluated i.e., i <= 10 which results into true.

3. Since, the test expression is true, the body of the loop i.e., System.out.println(" "+i) is executed which prints the current value of i on the same line.

4. After executing the loop body, the update expression i.e. ++i is executed which increments the value of i.

5. After the update expression is executed, the test-expression is again evaluated. If it is true,

the sequence is repeated from step no 3, otherwise the loop terminates.

e.g.2. Write code using for loop to print even numbers from 1 to 10.

2, 4 , 6 , 8 , 10

```
for(int i =2 ; i <= 10 ; i=i+2)  {
    System.out.print(" "+ i);
  }
```

The output will be : 2 4 6 8 10

e.g. 3 Write code using for loop to print odd numbers from 1 to 10.

1 , 3 , 5 , 7 , 10

```
for(int i =1 ; i <= 10 ; i=i+2)  {
    System.out.print(" "+ i);
  }
```

The output will be :1 3 5 7 9

When series to be displayed is ascending series, then <= operator is used in test expression.
When series to be displayed is descending series, then >= operator is used in test expression.

e.g.4. Write code using for loop to print numbers from 10 to 1

```
for(int b =10 ; b >= 1 ; b--)  {
    System.out.print(" "+ b);
  }
```

The output will be :10 9 8 7 6 5 4 3 2 1

e.g. 5 Write code using for loop to print the even numbers between 10 and 1.

10 8 6 4 2

```
for(int p =10 ; p >=2 ; p=p-2)  {
        System.out.print(" "+ i);
      }
```

e.g. 5 Write code using for loop to print the odd numbers between 10 and 1.

9 7 5 3 1

```
for(int p =9 ; p >=1 ; p=p-2)  {
        System.out.print(" "+ p);
      }
```

e.g. 6 Write code using for loop to print the sum of series 1+2+3+4+…….+10.

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

```
                    int sum = 0;
for(int a =1 ; a <= 10 ; a=a+1)  {
        sum = sum + a ;
      }
System.out.print(" "+ sum);
```

The output will be : 55

e.g. 7 Write code using for loop to print the sum of EVEN numbers between 1 and 10.

2 + 4 + 6 + 8 + 10

```
                    int sum = 0;
for(int k =2 ; k <= 10 ; k=k+2)  {
        sum = sum + i ;
      }
System.out.print(" "+ sum);
```

The output will be : 30

e.g. 8 Write code using for loop to print the sum of ODD numbers between 1 and 10.

1 + 3 + 5 + 7 + 9

```
int sum = 0;
for(int j =1 ; j <= 9 ; j=j+2)  {
        sum = sum + j ;
    }
 System.out.print(" "+ sum);
```

The output will be : 25

e.g. 9. for loop to print factorial of a number n.

1! = 1

2! = 2 * 1 = 2

3! = 3 * 2 * 1 = 6

4!= 4 * 3 * 2 * 1 = 24

5! = 5 * 4 * 3 * 2 * 1 = 120

```
int f=1;
for(int a =n ; a >= 1 ; a=a-1)  {
        f = f * i ;
    }
System.out.print("The factorial is "+ f);
```

e.g.10. for loop to print odd number between 50 to 100.

e.g. 11. for loop to print the sum of following series:

1 + 1/4 + 1/7 + 1/10 + 1/13 + 1/16 + 1/19 + 1/22 + 1/25

Variations in for loop

1. Multiple initialization and Update expression

- A for loop may contain multiple initialization and multiple update expressions. These multiple expression must be separated by commas.

```
int i , sum ;

for(i=1, sum =0 ; i < = n ; sum+=i , i++)
```

Multiple Initialization expression

Multiple Update expression

2. Optional Expression

- In a for loop, ***initialization expression*** , ***test expression*** and ***update expression*** are optional, i.e. you can skip any or all of these expressions.

e.g.

Initialization expression skipped

```
int i =1 , sum = 0 ;

for(    ;  i <=20 ; i= i + 1 )

System.out.println(" "+i) ;
```

3. Infinite loop

- An infinite loop can be created by omitting the test expression as shown below :

```
for(j = 25 ;    ; -- j )

    System.out.println("An infinite loop");
```

- Similarly, the following for loop is also an infinite loop.

```
for(      ;      ;      )

        System.out.println("Endless loop");
```

while loop

- The while loop is an entry controlled loop. The syntax of a while loop is:

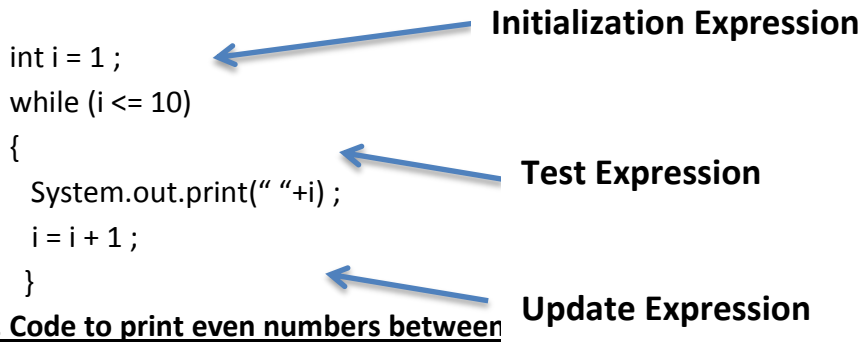
```
initialization expression ;
while(test expression)
{
   loop- body
   Update expression ;
  }
```

where loop body may contain a single or multiple statement.

The loop ***iterates*** while the **expression** evaluates to **true.** When the **expression** becomes **false**, the program control passes to the line after the loop body.

- In a while loop, a ***loop variable*** should be used initialised before the loop begins. The ***loop variable*** should be updated inside the body of while loop.

e.g.1. Code to print series from 1 to 10 using while loop.



e.g.2. Code to print even numbers between

2 4 6 8 10

```
int  i = 2 ;
while ( i <= 10)
{
    System.out.print(" "+i) ;
    i = i + 2 ;
 }
```

e.g.3. Code to print odd numbers between 10 to 1 using while loop.

9 7 5 3 1

```
int  i = 9 ;
  while ( i >= 1)
  {
      System.out.print(" "+i) ;
      i = i - 1 ;
   }
```

e.g.3. Code to print odd numbers between 1 to 10 using while loop.

1 3 5 7 9

```
int  i = 1 ;
  while ( i <= 9)
  {
       System.out.print(" "+i) ;
       i = i + 2 ;
   }
```

e.g. 4. Write code using while loop to print the sum of series 1+2+3+4+…….+10.

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

```
 int k = 1 , sum =0 ;
 while( k <= 10)
  {
     sum = sum + k ;
     k = k + 1 ;
   }
    System.out.print(" "+sum) ;
```

<u>e.g. 5.</u> while loop to print factorial of a number n.

```
 int  i = n , f =1 ;
 while( i >= 1)
 {
      f  = f * i  ;
      i = i – 1 ;
    }
   System.out.print("The factorial is "+ f);
```

<u>Variations in while loop</u>

- A while loop can be ***infinite*** if you forget to write the update expression inside its body.

e.g.

```
  int  i = 2 ;
  while ( i <= 10)
  {
       System.out.print(" "+i) ;
   }
```

This loop is not having update expression, it will print 1 infinite number of times.

do while loop

- do while is an ***exit controlled loop*** i.e. it evaluates its **test expression** at the bottom of the loop after executing its loop body statements. This means that a **do while loop always executes at least once.**

- **In do while** loop, the loop body is executed at least once, no matter what the initial state of **test-expression.**

Syntax of do while loop:

```
initialization expression ;
do {
      body of loop
      update expression ;
    } while (test expression) ;
```

e.g.1. Code to print series from 1 to 10 using do while loop.

```
int k =1 ;
do {
   System.out.print(" "+ k) ;
   k = k + 1;
  } while ( k <= 10);
```

Initialization expression

Update expression

Test expression

Upon execution, the above code will yield :

1 2 3 4 5 6 7 8 9 10

e.g. 2 Write code using do while loop to print the sum of EVEN numbers between 1 and 10.

2 + 4 + 6 + 8 + 10

```
int l = 2 , sum =0 ;
do {
   sum = sum + l ;
   l = l + 2 ;
  } while ( l <= 10)
```

e.g. 3. Code to display count from 10 to 0 and then display "HAPPY LOOPING".

e.g. 4. Loop to display the following series upto 10 terms :

10 13.5 17 20.5 .....

e.g 5. Program to calculate and print the sums of even and odd integers of the first n natural numbers using a while loop.

Jump statements

- Jump statements unconditionally transfer program control.
- Java has three jump statements : **return** , **break** , **continue**

break statement

- This statement enables a program to skip over part of the code.
- It can be used with loops (***for, while , do while***), and selection statements (**if , switch**)
- Execution resumes at the statement immediately following the body of the terminated statement.

e.g.

```
for (int i = 2 ; i <= 10 ; i+=2)
{
   if( i == 4)
        break ;
   else
        System.out.print(" "+i);
}
```

The output of above code is : 2

continue statement

- continue statement also helps in skipping over part of the code.
  But instead of forcing termination, it forces the next ***iteration*** of the loop to take place, skipping any code in between.

e.g.

```
for (int i = 2 ; i <= 10 ; i+=2)
{
   if( i == 4)
        continue ;
   else
        System.out.print(" "+i);
}
```

The output of above code is : 2 6 8 10