UNIT-I

Introduction to Java: Overview – Features – Fundamental OOPS concepts – JDK – JRE – JVM -Structure of a Java program – Data types – Variables – Arrays – Operators –Keywords – Naming Conventions – Control statements, Type conversion and Casting – Scanner – String – equals(), equalsIgnoreCase(), length().

Unit – I

Introduction to Java

Overview:

Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java's object-oriented features were influenced by C++.This language was initially called "Oak," but was renamed "Java" in 1995.

James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems begin developing Java.

What is Java?

- A high-level, object-oriented programming language.
- Developed by Sun Microsystems (now owned by Oracle Corporation).
- Released in 1995.

Features:

- **Object-Oriented:**
  Java is based on the concept of objects and classes, which makes it easy to organize and reuse code.

- **Platform Independent:**
  Java programs can run on any device that has a Java Virtual Machine (JVM) installed, regardless of the device's operating system or hardware.

- **Simple:**
  Java is easy to learn and understand, with a syntax similar to other programming languages like C and C++.

- **Secure:**

  Java has built-in- security features like data encryption, secure socket layers, and access control, making it a popular choice for developing secure applications and protecting sensitive data.

- **Dynamic:**

  i. Java is considered to be more dynamic than C/C++ since it is designed to adapt to an evolving environment.
  ii. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

- **Interpreted and High performance:**

  i. Java allows you to create programs that can run on any type of computer by converting the program into a special kind of code called Java bytecode.
  ii. This bytecode can then be run on any computer that has the Java Virtual Machine (JVM) installed.

- **Multithreading:**

  Java supports multithreading and it allows developers to create programs that can run multiple tasks simultaneously, improving performance and efficiency.

- **Robust:**

  Java has a strong focus on reliability and durability, with features like memorymanagement and exception handling that help prevent common programming errors.

Fundamental OOPS concepts:

- ❖ Polymorphism
- ❖ Inheritance
- ❖ Encapsulation
- ❖ Abstraction
- ❖ Classes
- ❖ Object

Polymorphism:

- The ability of an object to take multiple forms.
- Method overriding and method overloading are types of polymorphism.
- Method Overriding:
    i. Run-time polymorphism.
- Method Overloading:
    i. Multiple methods with the same name but different parameters.
    ii. Compile-time polymorphism.

Inheritance:

- Inheritance is an important pillar of OOP.
- A mechanism for creating a new class based on an existing one.
- The new class (subclass) inherits properties and behavior from the existing class (superclass).

Encapsulation:

- It is defined as the wrapping up of data under a single unit.
- The concept of hiding implementation details and showing only necessary information.
- Data hiding and abstraction.

Abstraction:

- An essential element of Object oriented Programming is Abstraction.
- Showing only essential features and hiding non-essential details.
- In Java, Abstraction is achieved by Abstract classes and Interfaces.

Class:

- A class is a blueprint or template for creating objects.
- A Java class uses variables to define data fields and methods to define behaviors.

Object:

- An Object is an instance of a class.
- Memory is allocated only after object instantiation.
- An Object has both a state and behavior.
- The state defines the object, and the behavior defines what the object does.

Basic Terminologies:

1. **Superclass:**
    - The class whose features are inherited is known as superclass.
    - (base (or) parent class)

2. **Subclass:**
    - The class that inherits the other class is known as Subclass.
    - (derived (or) extended (or) child class)

3. **Reusability:**
    - Reusability is a mechanism which allows to reuse the fields and methods of the existing class while creating a new class.

JDK:

- JDK (Java Development Kit) is a software development package that includes the Java Runtime Environment (JRE), compiler, and tools for developing, testing, and running Java programs.
- In Simpler terms, JDK is a bundle of tools that allow you to:
    - Write Java code
    - Complile it
    - Run it
    - Test it
- JDK is essential for Java development, and it's a must-have for any Java Programmer!

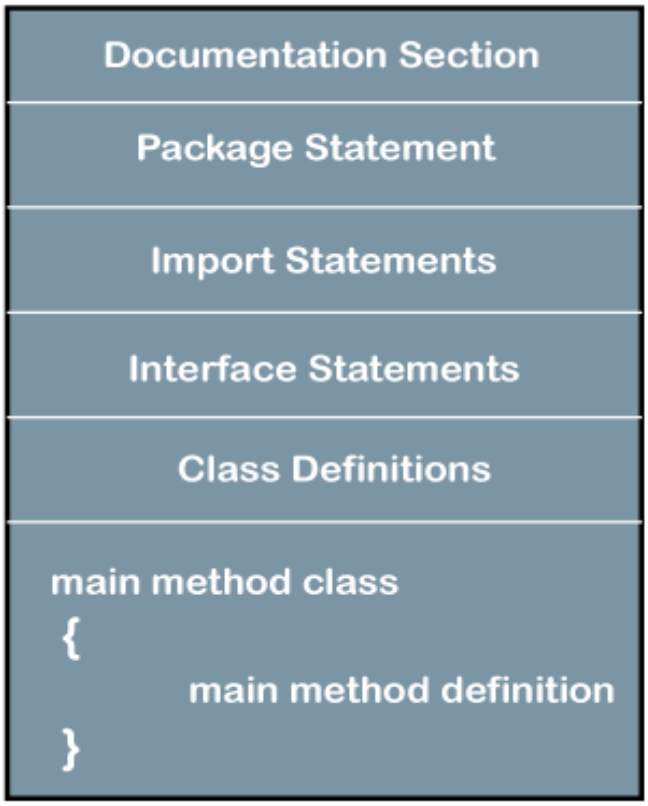
JRE:

- JRE (Java Runtime Environment) is a software package that includes the Java Virtual Machine (JVM), libraries, and utilities necessary to run Java programs, but not to develop them.
- In Simpler terms, JRE is a bundle that includes:
    - JVM (the engine that runs Java code)
    - Libraries (pre-built Java classes)
    - Utilities (tools for running Java programs)
- JRE is essential to run Java programs, but it doesn't include the compiler or other development tools.

JVM:

- The Java Virtual Machine is software that interprets Java bytecode
- Java programs executed in JVM
- The JVM is typically implemented as a run time interrupter
- Translate java bytecode instructions into object code
- JVM is the engine that runs Java programs, making Java a "write once, run anywhere" language!

Structure of a Java program:



Documentation section:

- It is a important but optional section in java, which include basic informations like author's name, date of creation and other descriptions.

  Single line comment - //
  Multi line comment - /...../
  Documentation comment - /*...../

Package declaration:

- It is also optional and placed right after the documentation section.
- Here we declare the package name in which the class is place and there can be only one package statement in java.

Import statements:

- The import statement represents the class stored in the other package.
- We use the import keyword to import the predefined classes and interfaces in a particular package.
- We can use multiple import statements.

Interface section:

- It is optional ie. we can use the keyword interface to create an interface if required.
- An interface is slightly difference from class as it contains only constants and method declarations and also it cannot be instantiated.

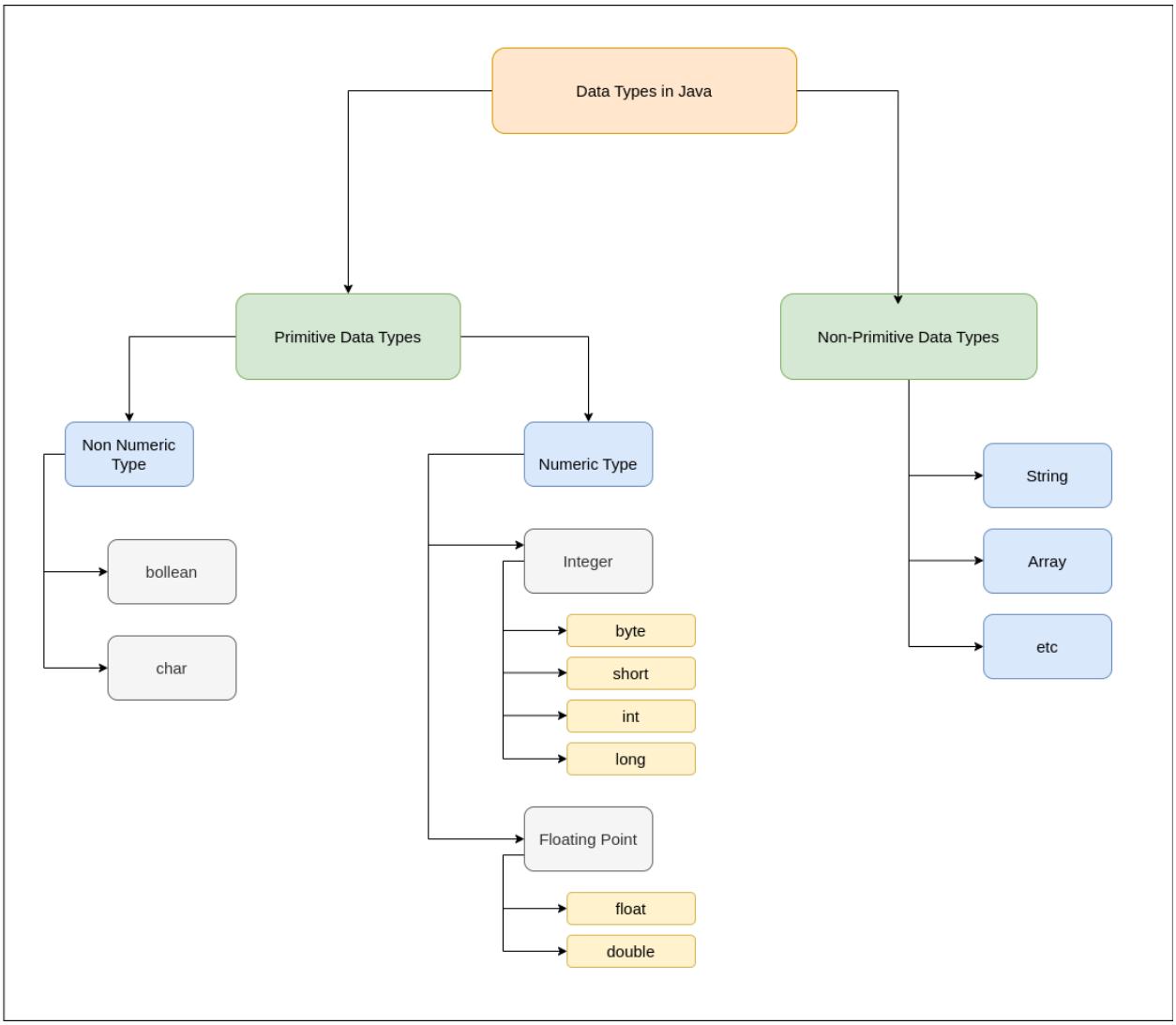
Class definition:

- It is a vital part as we cannot create any java program without class.
- There may be more than one class definition.
- It is declared using the keyword class.
- It contains information about user-defined methods, variables and constants.

Main method class:

- The execution of all Java programs starts from the main() method.
- In other words, it is an entry point of the class.
- It must be inside the class.
- Inside the main method, we create objects and call the methods.

DATA TYPES IN JAVA



Integers:

- In Java there are 4 integer types: byte, short, int and long.
- All of these are signed positive and negative values.
- Java does not support unsigned, positive only integers.

i) Byte:

- The smallest integer type is byte.
- This is signed 8-bit type that has a range from -128 to 127.
- It is declared by using the keyword byte.

- Variables of type byte are very useful while working with a stream of data.
- They are also useful when you are working with raw binary data that may not be directly compatible with Java's other built-in-types.
- **For example:**

  The following declares 2 byte variable called b and c ;

  byte b, c;

ii) Short:

- Short is signed 16-bit types.
- It has a range from -32,768 to 32,767.
- It is the least used Java type.
- **For example:**

```
short s;
short t;
```

iii) Int:

- The most commonly used integer type is int.
- It is a signed 32-bit type that has a range from -2,147,483,648 to 2,147,483,647.
- Int is commonly used to control loops and to index arrays.
- **For example:**

```
int a = 100000 ;
int b = -200000 ;
```

iv) Long:

- Long is a signed 64-bit type and is useful when an int type is not large enough to hold the desired value.
- The range of long is quite large.
- It useful when big, whole numbers are needed.
- **For example:**

```
long a = 100000L;
long b = -200000L;
```

Floating-Point Types:

- Floating-point numbers also known as real numbers, are used when evaluating expressions that require fractional precision.
- There are 2 kinds of floating-point types, float and double which represent single and double precision numbers respectively.

i) Float:

- The type float specifies a single-precision value that uses a 32 bits of storage.
- Variables of type float are used when a fractional component is needed, but it doesn't require a large degree of precision.
- For example float can be used to represent dollars and cents.
- **Example: float hightemp, lowtemp;**

ii) Double:

- Double precision, denoted by the double keyword, uses 64 bits to store a value.
- When a large degree of precision is needed, double is used.
- **Example: double d1 = 12.3**

Characters:

- The char data type is a single 16-bit Unicode character with the size of 2 bytes.
- The range of char is 0 to 65,536.
- There are no negative chars.
- **Example: char letter A = 'A'**

Booleans:

- Java has a primitive type called Booleans for logical values.
- A Boolean expression returns a Boolean value: True or False.

Java Keywords:

- There are 50 keywords currently defined in the Java Language.
- These Keywords cannot be used as names for a variable, class or method.
- The keywords const & goto are reserved but not used.
- In addition, Java reserves true, false and null also.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

TABLE 2-1 Java Keywords

Variables:

- The variables are the basic unit of storage in Java Program.
- A variable is defined by the combination of an identifier, a type and an optional initializer.

Declaring a variable:

- In Java, all variables must be declared before they can used.
- **Syntax:**

  type identifier [= value] [, identifier [= value] ...];
- type - Java's atomic types, or the name of a class or interface.
- identifier - name of the variable.
- **Example:**

```
int a, b, c;                    // declares three integers, a, b, and c.
int d = 3, e, f = 5;
```

Scope and lifetime of the variables:-

- A block defines a scope.
- Thus, each time you start a new block, you are creating a new scope.
- A scope determines what objects are visible to other parts of your program.
- It also determines the lifetime of those objects.
- There are two categories of scope they are global and local.
- The two major scopes are those defined by a class and those defined by
- a method.
- Scopes can be nested.

Dynamic Initialization:-

- Java allows variables to be dynamically initialized using any valid expression at declaration.

```
class DynInit
{
    public static void main(String args[])
    {
        double a = 3.0, b = 4.0;
        // c is dynamically initialized
        double c = Math.sqrt(a * a + b * b);
        System.out.println("Hypotenuse is " + c);
    }
}
```

- In the example program, the variables a and b are constants, while c is calculated dynamically using the Pythagorean theorem.
- The program computes the hypotenuse of a right triangle with sides a and b by using Math.sqrt(a * a + b * b).

Array:

- Java array is an object which contains elements of a similar data type.
- A specific element in an array is accessed by index.
- Arrays offer a convenient means of grouping related information.
- There are two types of array :
  i. Single dimensional array
  ii. Multi dimensional array

One Dimensional Array:- (Single dimension)

- An one-dimensional array is a list of like-typed variables.
- The general form of one-dimensional array declaration is
  type var_name[ ];
- type - the base type of the array
- The base type determines the data types of each element of the array.
- Arrays can be initialized when they are declared.
- The initialization of an array is a list of values separated by commas and surrounded by curly braces.
- The general form of one-dimensional array initialization is
  type var_name[ ] = { values of elements separated by commas } ;
- **Example:** int marks [ ] = { 90 , 85 , 75, 90 , 77 };

Example of One-dimensional array:

```
class Array
{
    public static void main(String args[ ])
    {
        int month_days[];
        month_days = new int[12];
        month_days[0] = 31;
        month_days[1] = 28;
        month_days[2] = 31;
        month_days[3] = 30;
```

```
        month_days[4] = 31;
        month_days[5] = 30;
        month_days[6] = 31;
        month_days[7] = 31;
        month_days[8] = 30;
        month_days[9] = 31;
        month_days[10] = 30;
        month_days[11] = 31;
        System.out.println("April has " + month_days[3] + " days.");
    }
}
```

Output:

April has 30 days.

Multi Dimensional Array:-

- Multi dimensional arrays are arrays of arrays.
- Each element of this array holds the reference of other arrays.
- The general form of multi-dimensional array declaration is

  type var_name [ ] [ ] ;

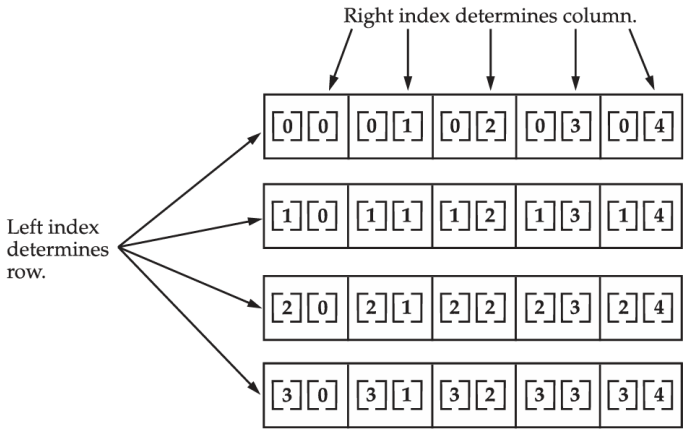


FIGURE 3-1 A conceptual view of a 4 by 5, two-dimensional array

- The initilization of multi-dimensional array can be done by
  int [ ] [ ] array = { { 1 , 2 } , { 3 , 4 } }

Example of Multi-dimensional Array:

```
class MDA
{
   public static void main(String[] args)
   {
       int [ ] [ ] arr = { { 1, 2 }, { 3, 4 } };
       for (int i = 0; i < 2; i++)
       {
           for (int j = 0; j < 2; j++)
           {
               System.out.print(arr[i][j] + "  ");
           }
           System.out.println();
       }
   }
}
```

Output:

```
1  2
3  4
```

OPERATORS IN JAVA

Arithmetic operators:

- The Arithmetic Operators are +, -, *, /, %
- The operand of arithmetic operator must be a number type.
- Floating point types can also be evaluated using arithmetic operators.

Operators	Meaning	Description
1. +	Addition or unary plus	Performs addition operation.
2. –	Subtraction or unary minus	Performs subtraction operation.
3. *	Multiplication	Performs multiplication operation.
4. /	Division	Performs division operation.
5. %	Modulo division (Remainder)	Performs remainder after division operation.

Example:

```
class Basicmath
{
        public static void main (string arg[])
        {
                int a = 20, b = 10;
                System.out.println("a: " +a);
                System.out.println("b: " +b);
                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 / b = " +(a / b));
        }
}
```

Output:

a: 20
b: 10
a + b = 30
a - b = 10
a * b = 200
a / b = 2

Modulus operator:

- Modulus Operator returns the remainder of a division operator.
- It can be applied to floating point types as well as integer type.

Example:

```
class modulus
{
	public static void main (string args [])
	{
		int x = 42;
		double y = 42.5;
		System.out.println (“X mod 10=” +X%10);
		System.out.println (“Y mod 10=” +Y%10);
	}
}
```

Output:

X mod 10 = 2
Y mod 10 = 2.25

Assignment operators:

- An operator which is used to store a value into a particular variable is called assignment operator in Java.
- These operators are used to assign values to a variable.
- Example: a = a+4 can be written as a += 4

Example :

```
class example
{
        public static void main (string args [])
        {
                int a = 1;
                int b = 2;
                int c = 3;
                a += 5;
                b *= 4;
                c += a*b;
                System.out.println (“a=” +a);
                System.out.println (“b=” +b);
                System.out.println (“c=” +c);
        }
}
```

Output:

```
a=6
b=8
c=51
```

Relational operator:

- Compares two values and takes decisions.
- Comparison can be done with the help of relational operator.
- Example: a < b or x < 20
- An expression containing a relational operator is termed as relational expression.

OPERATOR	MEANING
<	Is less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Is not equal to
==	Is equal to

Example:

```
class example
{
        public static void main (string args [])
        {
                int a = 5;
                int b = 10;
                if (a<b)
                        System.out.println ("The value of a is small");
                else
                        System.out.println ("The value of b is small");
        }
}
```

Output:

The value of a is small

Increment and Decrement operators:

- The operator ++ adds 1 to the operand.
- The operator -- subtracts 1 from the operand.
- Example: (i) ++a or a++
  (ii) --a or a--
- We can use this operator in for and while loops

Example :

```
class IncrementDecrement
 {
        public static void main(String args[])
        {
              int number = 10;
              System.out.println("Initial number: " + number);

              number++;
              System.out.println("After increment: " + number);

              number--;
              System.out.println("After decrement: " + number);
         }
}
```

Output:

```
Initial number: 10
After increment: 11
After decrement: 10
```

Ternary operator (Conditional Operator ?) :

- Java has a conditional operator that evaluates which of the two expression is evaluated.
- The result of the chosen expression is the result of the entire conditional operator.
- **Syntax :**

  condition ? expression 1 : expression 2
- If the conditional is true, expression 1 is evaluated, if it is false, expression 2 is evaluated,
- The conditional operator is similar to the if-else statement.
- The conditional operator is ternary because it requires three operands.

Example:

```
class Ternary
{
    public static void main(String[] args)
    {
        int x = 20;
        int y = 10;
        int z = (x > y) ? x : y;
        System.out.println("Greatest number: " +z);
    }
}
```

Output:

Greatest number: 20

Boolean Logical Operators:

- A Boolean expression returns a boolean value: true or false.
- Logical operators are used to determine the logic between variables or values.

OPERATOR	RESULT
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Example:

```
class BooleanLogical
{
	public static void main(String args[])
	{
		boolean a = true;
		boolean b = false;
		boolean c = a | b;
		boolean d = a & b;
		boolean e = a ^ b;
		boolean f = !a;
		System.out.println(" a = " + a);
		System.out.println(" b = " + b);
		System.out.println(" a|b = " + c);
		System.out.println(" a&b = " + d);
		System.out.println(" a^b = " + e);
		System.out.println(" !a = " + f);
	}
}
```

Output:

```
a = true
b = false
a|b = true
a&b = false
a^b = true
!a = false
```

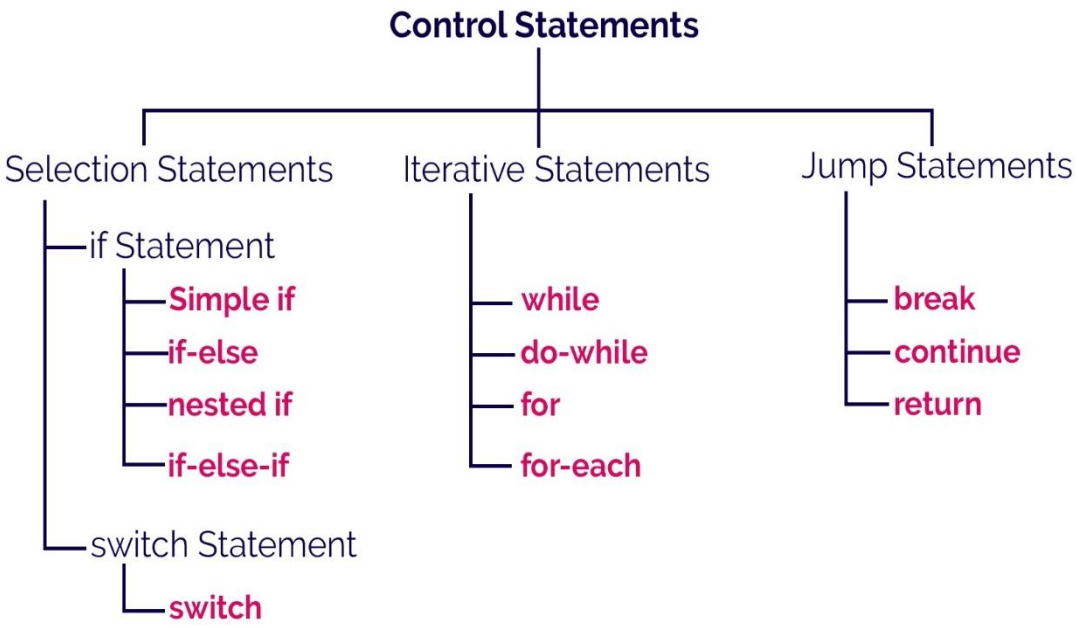
Bitwise Operators:

- Java defines several bitwise operators that can be applied to integer types, long, int, short, char and byte.
- These operators act upon the individual bits of their operands.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

JAVA CONTROL STATEMENTS

1) Decision Making statements (Selection Statements)
2) Looping Statements (Iterative Statements)
3) Jump Statements



1) Decision Making Statements:

- This statement allows you to control the flow of your programs execution using conditions known only during run time.
- There are 2 types of decision-making statements in java
- (i.e) if statement and switch statement

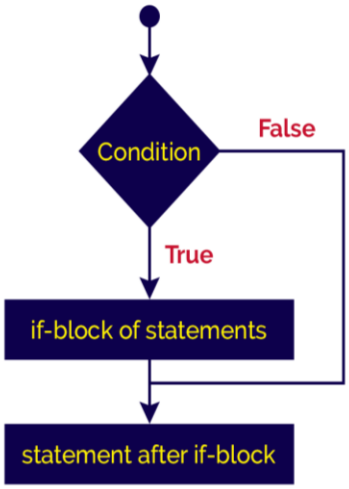
i) Simple if:

- If statement is the very simple decision making statement.
- It is used to decide whether a certain statement (or) Block of statements will be executed or not.
- If a certain condition is true then a block of statements is executed otherwise not.

Syntax

```
if(condition){
        if-block of statements;
        ...
}
statement after if-block;
...
```

Flow of execution



Example:

```
class Student
{
        public static void main (String args[])
        {
                int x = 10;
                int y = 12;
                if(x+y > 20)
                {
                        System.out.println("x+y is greater than 20");
                }
        }
}
```

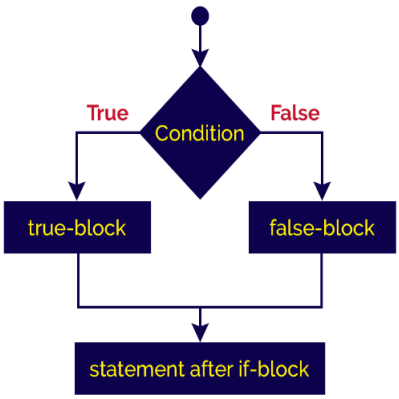
ii) If-else Statement:

- The if-else Statement uses another block of code. (i.e) else block.
- The else block is executed if the condition of the if-block is evaluated as false.

Syntax

```
if(condition){
        true-block of statements;
        ...
}
else{
        false-block of statements;
        ...
}
statement after if-block;
...
```

Flow of execution



Example:

```
class Student
{
        public static void main (String args[])
        {
                int x = 10;
                int y = 12;
                if (x+y < 10)
                {
                        System.out.println("x+y is less than 10");
                }
                else
                {
                        System.out.println("x+y is greater than 10");
                }

        }
}
```

iii) If-else-if ladder:

- The if-else-if statement contains the if-statement followed by multiple else-if statements.
- When the 'if' is true, the statement associated with that 'if' is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true then the final else statement, which is the default condition will be executed.

Syntax:

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

Example:

```
public static void main(String args[])
{
        Int i = 20;
        if (i==10)
                System.out.println("i is 10");
else if (i==15)
                System.out.println("i is 15");
else if (i==20)
                System.out.println("i is 20");
else
                System.out.println("i is not present");
}
```

Flow of execution:

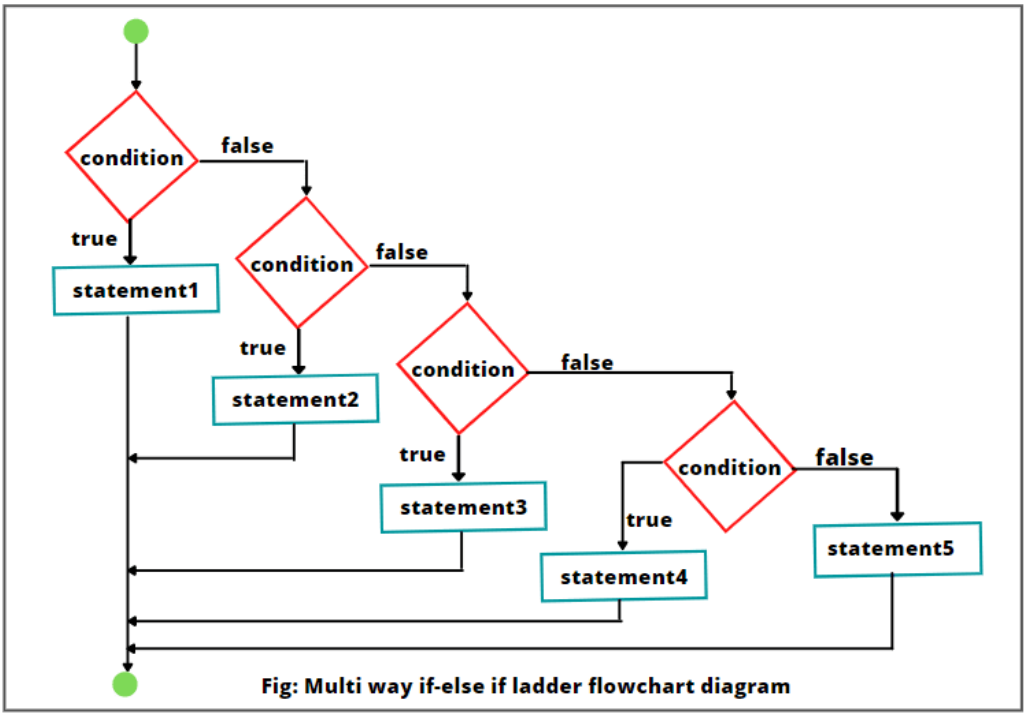


Fig: Multi way if-else if ladder flowchart diagram

iv) Nested if:

- In nested-if statement the if-statement can contain a if or if-else statement inside another if or else-if-statement.
- **Example:**

```
if(i == 10)
{
    if (j < 20) a = b;
            if (k > 100)
                    c = d;
             else a = c;
 }
else a = d;
```

v) Switch statement:

- The switch statement is a multiway branch statement.
- It helps to run different parts of your code based on what a specific value is.

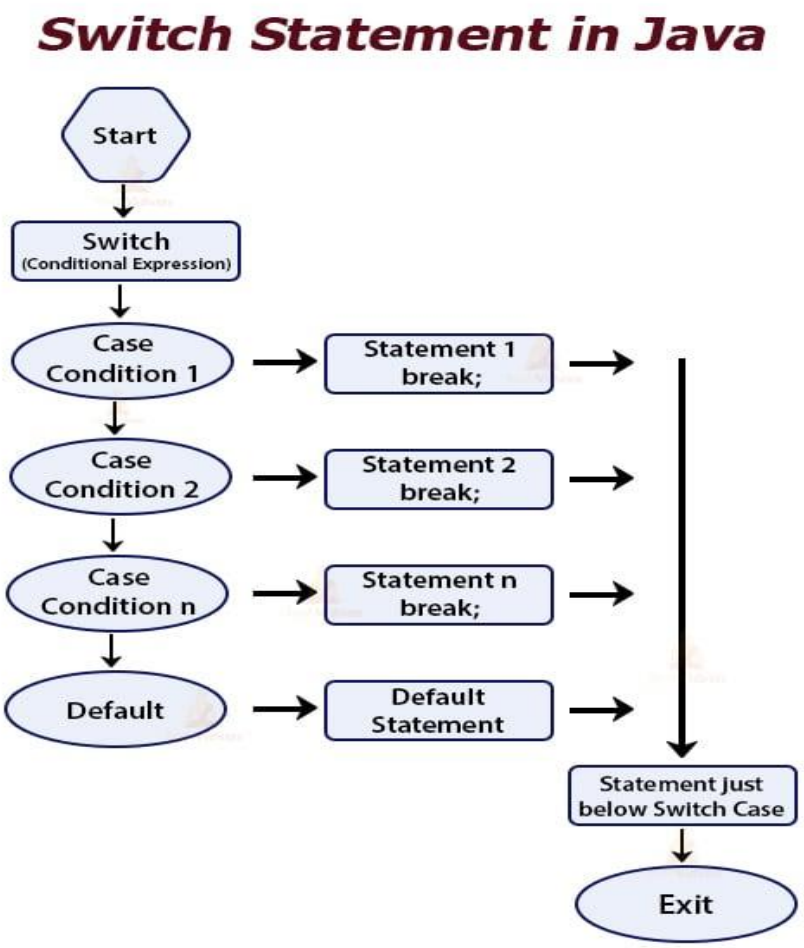
Syntax:

```
Switch (expression)
{
 case value 1:
          statement 1;
 break;
 case value 2:
          statement 2;
 break;
          .
          .
          .
case value N:
         statement N;
break;
default:
         statement default;
}
```

Example:

```
 public static void main (String args[])
{
         int num = 20;
         switch (num)
         {
                  case 5 :
                           System.out.println("It is 5");
                  break;
```

```
                case 10 :
                        System.out.println("It is 10");
                break;
                case 15 :
                        System.out.println("It is 15");
                break;
                case 20 :
                        System.out.println("It is 20");
                break;
                default :
                        System.out.println("Not present");
        }
}
```



2) Looping Statements (Iterative Statements)

- Java's Iteration statements are for, while and do-while.
- These statements are called loops.
- A loop repeatedly executes the same set of instruction until an end condition is met.

I. While loop:- (Entry control loop)

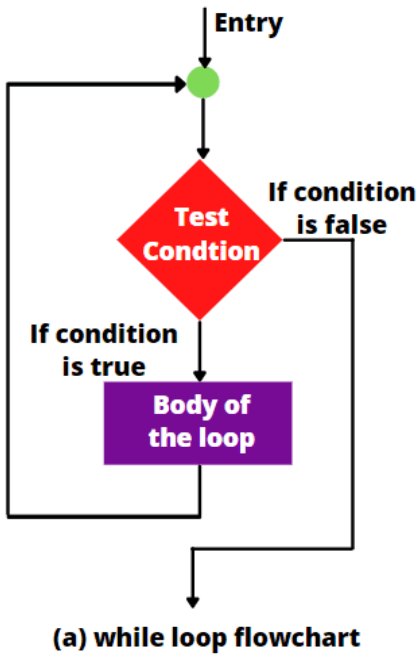
- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The while loop is like a repeating if statement.

Syntax:

```
While (boolean condition)
{
        loop statements…
}
```

Example:

```
public static void main (String args[])
{
        int i = 10;
        while(i<=10)
        {
                System.out.println(i);
                i++;
        }
}
```



(a) while loop flowchart

II. do while:- (Exit control loop)

- Do while loop is similar to while loop.
- The only difference is that it checks for condition after executing the statements.
- Do while loop is an example of exit control loop.

Syntax:

```
do
{
        Statements…
}
while (condition);
```

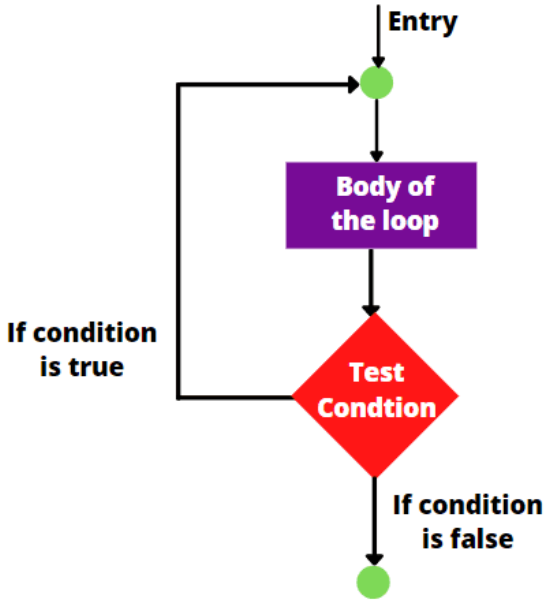


Fig: do while loop flowchart

Example:

```
public static void main (String args[])
{
        int i = 0 ;
        do
        {
                System.out.println(i);
                i++;
        }
        while(i<=10);
}
```

III. For loops :-

- for loop provides a clear way of writing the loop structure.
- Unlike a while loop, a for loop sets up the initialization, condition, and increment or decrement in one line, making it shorter and easier to debug.

Syntax:

```
for (initialization; condition; iteration)
{
    statements;
}
```

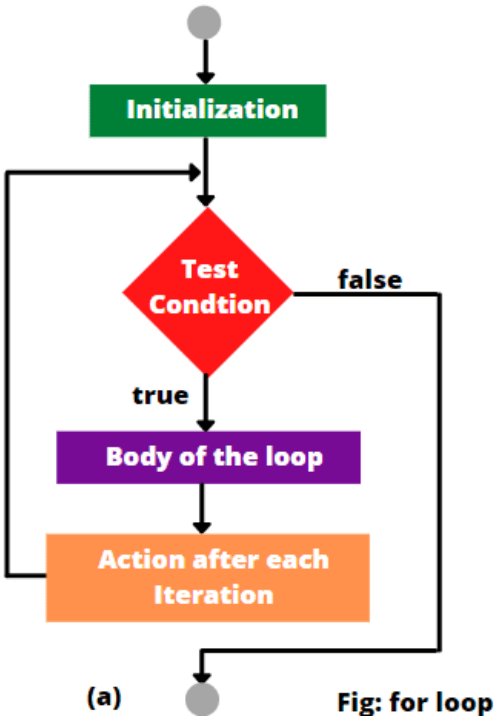


Fig: for loop

Example:

```
public static void main (String args[])
{
    for(int i=0; i<=10; i++)
    {
            System.out.println(i);
    }
}
```

3) Jump statements:-

- Java supports three jump statements: break, continue and return.
- These statements transfer control to another part of the program.

I. Break:-

- In java, the break statement has 3 uses:
  - i) Terminate a sequence in a switch statement
  - ii) To exit a loop
  - iii) Used as a "controlled" form of goto.
- **Example:**

```
public static void main (String args[])
{
        for(int i=0; i<=10; i++)
        {
                System.out.println(i);
                if(i==6)
                {
                        break;
                }
        }
}
```

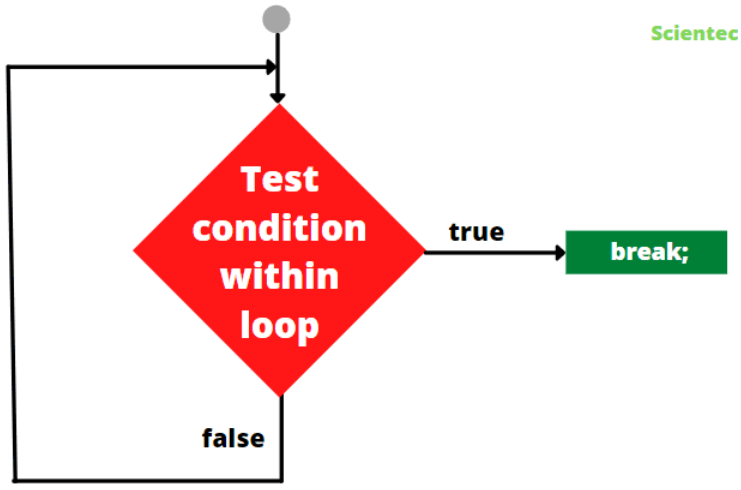


Fig: Flowchart of break statement

II. Continue :-

- Sometimes you need to skip the rest of the code in a loop and move to the next iteration.
- This means you want the loop to keep running, but skip the remaining code for the current loop cycle.
- It's like jumping to the end of the loop and starting the next cycle.
- The `continue` statement does this.

Example:

```
class Continue
{
        public static void main (String args[])
        {
                for(int i=0; i<10; i++)
                {
                        System.out.println(i +"");
                        if (i%2==0) continue;
                        System.out.println("");
                }
        }
}
```

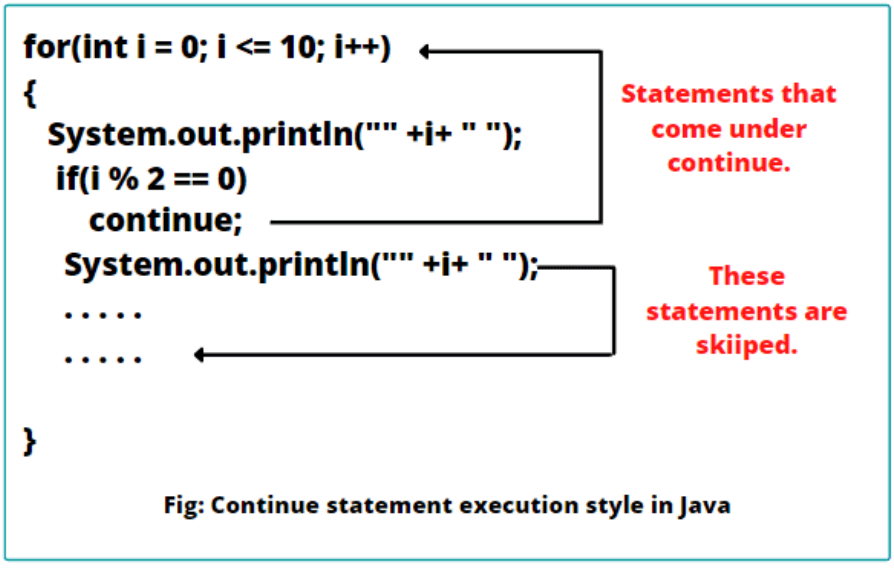


Fig: Continue statement execution style in Java

III. Return:-

- The `return` statement is used to exit a method.
- It sends control back to where the method was called from.

Example:

```
class Return
  {
  public static void main (String args[])
        {
              boolean t = true;
              System.out.println("Before the return.");
              if(t) return;
              System.out.println("This won't execute.");
        }
 }
```

Output:-

```
Before the return.
```

- As you can see, the final println( ) statement is not executed.
- As soon as return is executed, control passes back to the caller.

Java Naming Convention:

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
- It is not forced to follow. So, it is known as convention not rule.
- These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

Type conversion and casting :

- Type conversion in java refers to the process of converting one data type to another.
- There are two types of type conversion Automatic conversion and Explicit conversion.

Automatic conversion:

An automatic type conversion will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.
- It is also known as widening conversion.
- **Example:**

```
int x = 30;
float y ;
y = x;
```

Output : y=30.000000

Explicit conversion:

- When the destination type is smaller than the source type this method is used.
- It is also known as narrowing conversion.
- Cast is used to create conversion between two incompatible types.
- A cast is simply an explicit type conversion.
- General form is: (target-type) value
- **Example :**

```
int a;
byte b;
//…..
b = (byte) a;
```

Scanner:

- Scanner is a class in java.util package.
- It is used for obtaining the input of the primitive types like int, double, etc. and strings.
- Scanner class helps to take the standard input stream in Java.
- **Syntax :** Scanner obj_name = new Scanner (System.in)
- **Example:** Scanner scan = new Scanner(System.in)

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

Strings:

- The string is a sequence of characters.
- In Java, Objects of string are immutable which means a constant cannot be changed once created.
- Some of the commonly used string methods are

String name = "Dhoni";

i) name.length() : The length of a string is the number of characters that it contains. It returns string length

output : 5 (in this case)

ii) name.equals("dhoni") : Returns true if the given string is equal to "Dhoni", false otherwise. This comparison is case sensitive.

output : false [case sensitive].

iii) name.equalsIgnoreCase("dhoni") : Returns true if two strings are equal ignoring the case of characters.

output : true

* To compare two strings for equality use equals().

iv) name.toLowerCase() : Converts all the characters in a string from uppercase to lowercase.

output : dhoni

v) name.toUpperCase() : Converts all the characters in a string from lowercase to uppercase.

output : DHONI

vi) name.trim() : Returns a new string after removing all the leading and trailing spaces from the original string.

Example : String name = “ Dhoni “
Output : “Dhoni”

vii) name.substring(int start) : Returns a substring from start to the end.

name.substring(2) for the string "dhoni" would return "oni".

viii) name.substring (int start, int end) : Returns a substring from start index to the end index.

Start index is included and end index is excluded.

name.substring (1,3) ---> "ho"

ix) name.replace ('n','b') : Returns a new string after replacing n with b.

"Dhobi" is returned in this case.

x) name.startswith ("Dh") : Returns true if name starts with string "Dh".

True in this case.

xi) name.endswith ("i") : Returns true if name ends with string "i".

True in this case.

xii) name.charAt(2) : Returns character at a given index position.

"o" in this case.

xiii) name.index(i) : Returns the index of the given string.

4 in this case.

xiv) name.lastIndex ("h") : Returns the last index of the given string.

1 in this case.

xv) name.lastIndex ("h", 2) : Returns the last index of the given string before 2.

equals and equals Ignore case( )

- To compare two strings for equality use equals( ).
- It has a general form : **boolean equals (Object str).**
- str is the String object being compared with the invoking string object.
- It returns true if the strings contain the same charactgers in the same order, and false otherwuse.
- The comparison is case-sensitive.
- To perform a comparison that ignores case differences, call equalsIgnoreCase( ).
- When it compares two strings it considers A-Z to be the same as a-z.
- It has this general form: **boolean equalsIgnoreCase(String str).**
- str is the String object being compared with the invoking, string object.
- It too returns true if the strings contain the same characters in the same order, and false otherwise.
- Here is an example that demonstrates equals( ) and equalsIgnoreCase( ):

```
class equalsDemo
{
    public static void main (String args[ ])
    {
        String s1 = “Hello” ;
        String s2 = “Hello” ;
        String s3 = “Good-bye” ;
        String s4 = “HELLO” ;

        System.out.println(s1 +“equals” +s2 +”→” +s1.equals(s2));
        System.out.println(s1 +“equals” +s3 +”→” +s1.equals(s3));
        System.out.println(s1 +“equals” +s4 +”→” +s1.equals(s4));
        System.out.println(s1 +“equalsIgnoreCase” +s4 +”→”
        +s1.equalsIgnoreCase(s4));
    }
}
```

Output:-

Hello equals Hello → true
Hello equals Good-bye → false
Hello equals HELLO → false
Hello equalsIgnoreCase HELLO → true