**Java** is a [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [class-based](https://en.wikipedia.org/wiki/Class-based_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) [programming language](https://en.wikipedia.org/wiki/Programming_language) that is designed to have as few implementation [dependencies](https://en.wikipedia.org/wiki/Dependency_(computer_science)) as possible. It is a [general-purpose](https://en.wikipedia.org/wiki/General-purpose_language) programming language intended to let [programmers](https://en.wikipedia.org/wiki/Programmer) *write once, run anywhere* ([WORA](https://en.wikipedia.org/wiki/Write_once,_run_anywhere)),[[16]](https://en.wikipedia.org/wiki/Java_(programming_language)#cite_note-16) meaning that [compiled](https://en.wikipedia.org/wiki/Compiler) Java code can run on all platforms that support Java without the need to recompile.[[17]](https://en.wikipedia.org/wiki/Java_(programming_language)#cite_note-design_goals-17) Java applications are typically compiled to [bytecode](https://en.wikipedia.org/wiki/Java_bytecode) that can run on any [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) regardless of the underlying [computer architecture](https://en.wikipedia.org/wiki/Computer_architecture). The [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) of Java is similar to [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B), but has fewer [low-level](https://en.wikipedia.org/wiki/Low-level_programming_language) facilities than either of them. The Java runtime provides dynamic capabilities (such as [reflection](https://en.wikipedia.org/wiki/Reflective_programming) and runtime code modification) that are typically not available in traditional compiled languages.

Java gained popularity shortly after its release, and has been a very popular programming language since then.[[18]](https://en.wikipedia.org/wiki/Java_(programming_language)#cite_note-18) Java was the third most popular programming language in 2022 according to [GitHub](https://en.wikipedia.org/wiki/GitHub).[[19]](https://en.wikipedia.org/wiki/Java_(programming_language)#cite_note-19) Although still widely popular, there has been a gradual decline in use of Java in recent years with [other languages using JVM](https://en.wikipedia.org/wiki/List_of_JVM_languages) gaining popularity.[[20]](https://en.wikipedia.org/wiki/Java_(programming_language)#cite_note-:0-20)

Java was originally developed by [James Gosling](https://en.wikipedia.org/wiki/James_Gosling) at [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems). It was released in May 1995 as a core component of Sun's [Java platform](https://en.wikipedia.org/wiki/Java_(software_platform)). The original and [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) Java [compilers](https://en.wikipedia.org/wiki/Compiler), virtual machines, and [class libraries](https://en.wikipedia.org/wiki/Library_(computing)) were originally released by Sun under [proprietary licenses](https://en.wikipedia.org/wiki/Proprietary_license). As of May 2007, in compliance with the specifications of the [Java Community Process](https://en.wikipedia.org/wiki/Java_Community_Process), Sun had [relicensed](https://en.wikipedia.org/wiki/Software_relicensing) most of its Java technologies under the [GPL-2.0-only](https://en.wikipedia.org/wiki/GNU_General_Public_License) license. [Oracle](https://en.wikipedia.org/wiki/Oracle_Corporation) offers its own [HotSpot](https://en.wikipedia.org/wiki/HotSpot_(virtual_machine)) Java Virtual Machine, however the official [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) is the [OpenJDK](https://en.wikipedia.org/wiki/OpenJDK) JVM which is free open-source software and used by most developers and is the default JVM for almost all Linux distributions.

As of March 2024, [Java 22](https://en.wikipedia.org/wiki/Java_version_history) is the latest version. Java 8, 11, 17, and 21 are previous LTS versions still officially supported.

Java features

* Simple
* Platform independent
* Architectural neutral
* Portable
* Multi threading
* Distributed
* Networked
* Robust
* Dynamic
* Secured
* High performance
* Interpreted
* Object Oriented Programming Language

Simple:

JAVA is simple because of the following factors:

o JAVA is free from pointers hence , So less development time and less  
 execution time [whenever we write a JAVA program we write without pointers and  
 internally it is converted into the equivalent pointer program].

o Rich set of API (application protocol interface) is available to develop any complex  
 application.

o The software JAVA contains a program called garbage collector which is always used to  
 collect unreferenced (unused) memory location for improving performance of a JAVA program. [Garbage collector is the system JAVA program which runs in the background along with regular JAVA program to collect unreferenced memory locations by running at periodical interval of times for improving performance of JAVA applications.

o JAVA contains user friendly syntax’s for developing JAVA applications.

· Platform Independent:  
 A program or technology is said to be platform independent if and only if which can run on  
 all available operating systems.  
 The languages like C, Cpp are treated as platform dependent languages.

· When we write a C or Cpp program on dos operating and if we try to transfer that program to Unix operating system, we are unable to execute

· The language like JAVA will have a common data types and the common memory spaces on all operating systems and the JAVA software contains the special programs which converts the format of one operating system to another format of other operating system. Hence JAVA language is treated as platform independent language.

ARCHITECTURE NEUTRAL

· A language or technology is said to be architectural neutral which can run on any available processors in the real world.

· The languages like C, Cpp are treated as architectural dependent.

· The language like JAVA can run on any of the processor irrespective of their architecture and vendor.

portable

· A portable language is one which can run on all operating systems and on all processors irrespective their architectures and providers.

· The languages like C, Cpp are treated as non- portable languages whereas the language JAVA is called portable language.

· WRITE ONCE RUN ANYWHERE

MULTI THREADED

FLOW OF CONTROL IS KNOWN AS THREAD

· A multi threaded program is one in which there exists multiple flow of controls i.e., threads.

· A program is said to be multi threaded program if and only of there exists n number of sub-  
 programs. For each and every sub-program there exists a separate flow of control. All such flow of controls are executing concurrently. Such flow of controls is known as threads. Such type of applications is known as multi threading applications.

· Whenever we write a JAVA program there exists by default two threads. They are foreground/child thread and background/main/parent thread.

· A foreground thread is one which always executes user defined sub-programs. In a JAVA program there is a possibility of existing n number of foreground threads.

· A background thread is one which always monitors the status of foreground thread. In each and every JAVA program there exists only one background thread.

· Hence background thread will be created first and later foreground thread will be created.

DISTRIBUTED

· A service is a said to be a distributed service which runs in multiple servers and that service and can be accessed by n number of clients across the globe.

**Dynamic**

· Java is more dynamic compared to C and C++.

· It can adapt to its evolving environment.

· It allows programmers to dynamically link new class libraries, objects, and methods.

· Java programs can have a large amount of run-time information that can be used to resolve accesses to objects.

· Robust

· Robustness is the capacity of a computer system to handle the errors during execution and manage the incorrect input of data.

· Java is robust because it utilizes strong **memory management**.

· There is **automatic garbage collection** in Java which runs on the Java Virtual Machine to eliminate objects which are not being accepted by a Java application anymore.

· There are **type-checking mechanisms** and **exception-handling** in Java.

· All these features make Java robust.

Important application areas of java

**Enterprise Applications**:

* Java EE (Enterprise Edition), now Jakarta EE, is widely used for building robust and scalable enterprise applications. It supports various enterprise features such as transaction management, security, and web services.

**Web Development**:

* Java is extensively used for server-side web development. Frameworks like Spring MVC, JavaServer Faces (JSF), and Play Framework provide powerful tools for building dynamic and scalable web applications.

**Mobile Applications**:

* Android, the world's most popular mobile operating system, uses Java as its official language for Android app development. Android Studio, the official IDE, supports Java and Kotlin for developing Android applications.

**Desktop GUI Applications**:

* Java provides GUI libraries such as Swing and JavaFX for developing cross-platform desktop applications with rich graphical user interfaces. Applications like IDEs (e.g., IntelliJ IDEA), scientific tools, and business applications are often built using Java.

**Big Data**:

* Java is widely used in the Big Data ecosystem. Apache Hadoop, Apache Spark, and Apache Kafka are examples of Big Data frameworks implemented in Java. Java's scalability, performance, and ability to handle large datasets make it suitable for Big Data processing.

**Financial Applications**:

* Java is popular in the financial industry for developing trading applications, risk management systems, and banking applications. Its reliability, security features, and compatibility with existing systems make it a preferred choice for financial institutions.

**Scientific and Mathematical Computing**:

* Java libraries like Apache Commons Math and JScience are used for scientific computing, numerical analysis, and simulations. Java's support for complex mathematical operations and its multi-threading capabilities are beneficial in scientific applications.

**Cloud-Based Applications**:

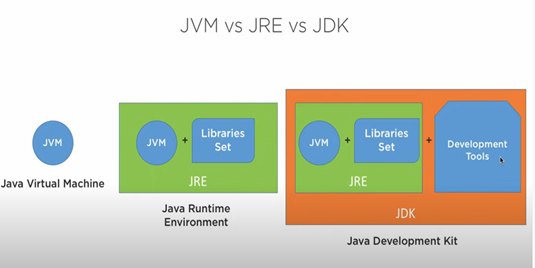
* Java is widely adopted for building cloud-native applications. Java frameworks such as Spring Boot facilitate the development of microservices architecture, which is common in cloud computing environments. Platforms like Kubernetes support Java applications seamlessly.

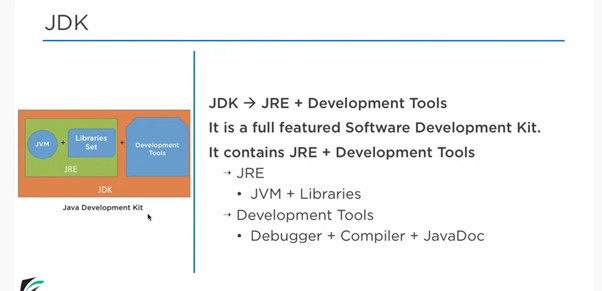
**Internet of Things (IoT)**:

* Java is used in IoT applications, particularly in embedded systems and IoT gateways. Java ME (Micro Edition) provides a lightweight runtime environment for IoT devices. IoT platforms like Eclipse IoT and Kura are built using Java to enable connectivity and management of IoT devices.

**Educational Tools and Learning Platforms**:

* Java's simplicity, platform independence, and strong community support make it suitable for educational purposes. It is widely used in teaching programming concepts and developing educational software, including learning management systems (LMS) and educational platforms.





**JDK**

**· Java Development Kit or JDK is the core component of Java Environment and provides all the tools, executables, and binaries required to compile, debug, and execute a Java Program.**

**· JDK is a platform-specific software and that’s why we have separate installers for Windows, Mac, and Unix systems.**

**· We can say that JDK is the superset of JRE since it contains JRE with Java compiler, debugger, and core classes.**

**JDK Development Tools**

**JDK (Java Development Kit) includes development tools and utilities for Java developers.**

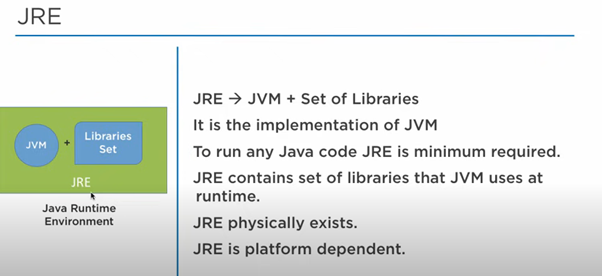
**These tools, such as**

**· Java compiler (javac),**

**· debugger (jdb), and**

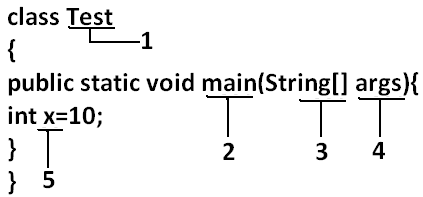
**· documentation generator (javadoc),**

**are used for compiling, debugging, and documenting Java programs.**

****

**Identifiers:-**

**A name in java program is called identifier. It may be class name, method name, variable name and label name.  
  
 Example:**

****

**Rules to define java identifiers:**

**Rule 1: The only allowed characters in java identifiers are:**

**1) a to z**

**2) A to Z**

**3) 0 to 9**

**4) \_ (underscore)**

**5) $**

**Rule 2: If we are using any other character we will get compile time error.**

**Example:**

**1) total\_number-------valid**

**2) Total#------------------invalid**

**Rule 3: identifiers are not allowed to starts with digit.**

**Example:**

**1) ABC123---------valid**

**2) 123ABC---------invalid**

**Rule 4: java identifiers are case sensitive up course java language itself treated as case sensitive language.**

**Example:**

**class Test{**

**int number=10;**

**int Number=20;**

**int NUMBER=20; we can differentiate with case.**

**int NuMbEr=30;**

**}**

**Rule 5: There is no length limit for java identifiers but it is not recommended to take more than 15 lengths.**

**Rule 6: We can't use reserved words as identifiers.**

**Example:**

**int if=10; --------------invalid**

**Rule 7: All predefined java class names and interface names we use as identifiers.**

**Example 1:**

**class Test**

**{**

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

**int String=10;**

**System.out.println(String);**

**}}**

**Output:**

**10**

**Example 2:**

**class Test**

**{**

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

**int Runnable=10;**

**System.out.println(Runnable);**

**}}**

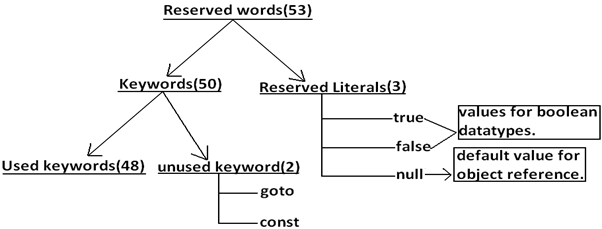
**Output:**

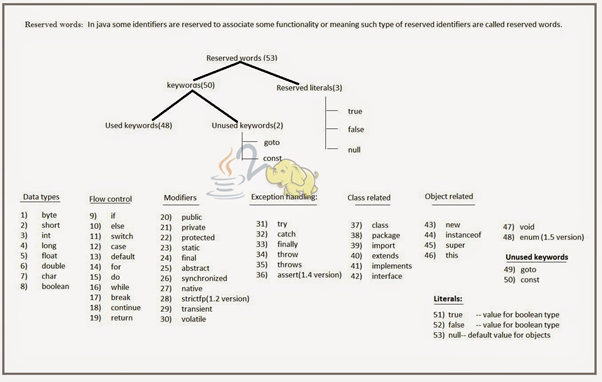
**10**

**Even though it is legal to use class names and interface names as identifiers but it is not a good programming practice.**

**Reserved words:**

**In java some identifiers are reserved to associate some functionality or meaning such type of reserved identifiers are called reserved words.  
  
 Diagram:**

****

****

**Java Data types & Literals**

**What is a data type ?**

**· A data type, in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.**

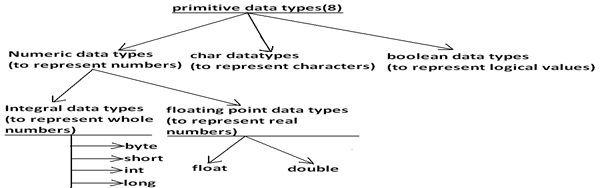
**· A string, for example, is a data type that is used to classify text and an int is a data type used to classify whole numbers.**

**Java Data types & Literals**

**What is a data type ?**

**· A data type, in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.**

**· A string, for example, is a data type that is used to classify text and an int is a data type used to classify whole numbers.**

****

**Java is Strongly typed language**

**· Every variable has a type and every expression has a type.**

**· All data types are strictly defined**

**· Every assignment will be checked by the compiler for the type compatibility**

**· Reason why Java is considered strongly typed programming language**

**Data types in java**

**----------------------------**

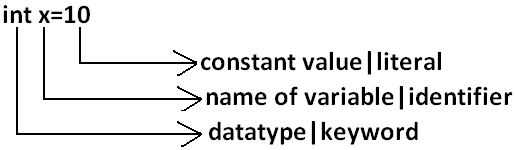
**Summary of java primitive data type:**

| **data type** | **Size** | **Range** | **Corresponding Wrapper class** | **Default value** |
| --- | --- | --- | --- | --- |
| **byte** | **1 byte** | **-27 to 27-1(-128 to 127)** | **Byte** | **0** |
| **short** | **2 bytes** | **-215 to 215-1 (-32768 to 32767)** | **Short** | **0** |
| **int** | **4 bytes** | **-231 to 231-1 (-2147483648 to 2147483647)** | **Integer** | **0** |
| **long** | **8 bytes** | **-263 to 263-1** | **Long** | **0** |
| **float** | **4 bytes** | **-3.4e38 to 3.4e38** | **Float** | **0.0** |
| **double** | **8 bytes** | **-1.7e308 to 1.7e308** | **Double** | **+** |
| **boolean** | **Not applicable** | **Not applicable(but allowed values true|false)** | **Boolean** | **false** |
| **char** | **2 bytes** | **0 to 65535** | **Character** | **0(represents blank space)** |

**Floating Point Data types:**

| **Float** | **double** |
| --- | --- |
| **If we want to 5 to 6 decimal places of accuracy then we should go for float.** | **If we want to 14 to 15 decimal places of accuracy then we should go for double.** |
| **Size:4 bytes.** | **Size:8 bytes.** |
| **Range:-3.4e38 to 3.4e38.** | **-1.7e308 to1.7e308.** |
| **float follows single precision.** | **double follows double precision.** |

**Data Literals**

**Any constant value which can be assigned to the variable is called literal.  
 Example:  
  
**

**Integral Literals:**

**For the integral data types (byte, short, int and long) we can specify literal value in the following ways.  
  
 1) Decimal literals: Allowed digits are 0 to 9.  
 Example: int x=10;  
  
 2) Octal literals: Allowed digits are 0 to 7. Literal value should be prefixed with zero.  
 Example: int x=010;  
  
 3) Hexa Decimal literals:**

* **The allowed digits are 0 to 9, A to Z.**
* **For the extra digits we can use both upper case and lower case characters.**
* **This is one of very few areas where java is not case sensitive.**
* **Literal value should be prefixed with ox(or)oX.**

**Example: int x=0x10;  
  
 These are the only possible ways to specify integral literal.**

**Which of the following are valid declarations?**

**1. int x=0777; //(valid)**

**2. int x=0786; //C.E:integer number too large: 0786(invalid)**

**3. int x=0xFACE; (valid)**

**4. int x=0xbeef; (valid)**

**5. int x=0xBeer; //C.E:';' expected(invalid) //:int x=0xBeer; ^// ^**

**6. int x=0xabb2cd;(valid)**

**String literals:**

**Any sequence of characters with in double quotes is treated as String literal.  
  
 Example:  
 String s="Ashok"; (valid)**

**1.7 Version enhansements with respect to Literals :**

**The following 2 are enhansements**

**1. Binary Literals**

**Binary Literals :**

**For the integral data types untill 1.6v we can specified literal value in the following ways**

**1. Decimal**

**2. Octal**

**3. Hexa decimal**

**But from 1.7v onwards we can specified literal value in binary form also.  
 The allowed digits are 0 to 1.  
 Literal value should be prefixed with Ob or OB .**

**int x = 0b111;**

**System.out.println(x); // 7**

**Example program on literals**

**—------------------------------------**

**public class LiteralsDemo{**

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

**int a=10;// decimal form**

**int b=0b10;// binary form**

**int c=000; // octal form.**

**int d=0x10; // hexa decimal form.**

**System.out.println(a+"========="+b+"=========="+c+"======="+d);**

**}**

**}**

## **Type Cast Operator :**

**There are 2 types of type-casting**

**1. implicit**

**2. explicit**

### **implicit type casting :**

**int x='a';**

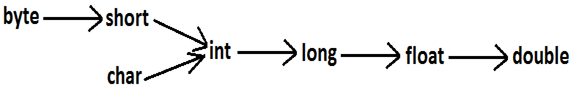
**System.out.println(x); //97**

**1. The compiler is responsible to perform this type casting.**

**2. When ever we are assigning lower datatype value to higher datatype variable then implicit type cast will be performed .**

**3. It is also known as Widening or Upcasting.**

**4. There is no lose of information in this type casting.**

**5. The following are various possible implicit type casting.  
 Diagram:  
**

**Note: Compiler converts int to double type automatically by implicit type casting.**

**Example program for implicit type casting**

**public class TemperatureConverter {**

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

**// Example of implicit type casting**

**int celsius = 25; // Temperature in Celsius**

**// Formula to convert Celsius to Fahrenheit**

**// celsius \* 9 / 5 + 32 involves implicit type casting from int to double**

**double fahrenheit = celsius \* 9 / 5 + 32;**

**// Print the result**

**System.*out*.println("Temperature in Celsius: " + celsius);**

**System.*out*.println("Temperature in Fahrenheit (implicit): " + fahrenheit);**

**}**

**}**

### **Explicit type casting:**

**1. Programmer is responsible for this type casting.**

**2. Whenever we are assigning bigger data type value to the smaller data type variable then explicit type casting is required.**

**3. Also known as Narrowing or down casting.**

**4. There may be a chance of lose of information in this type casting.**

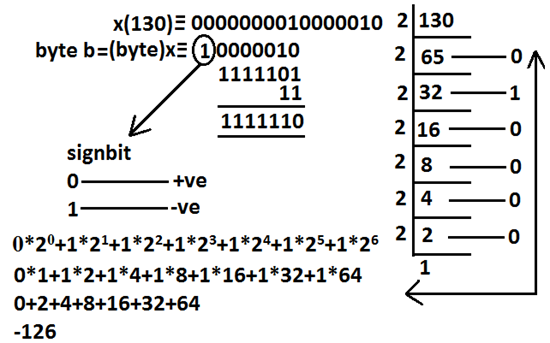
**5. The following are various possible conversions where explicit type casting is required.  
 Diagram:**

**Example :**

**int x=130;**

**byte b=(byte)x;**

**System.out.println(b); //-126**

**6**

**Example 2**

**int x=130;**

**byte b=x;**

**System.out.println(b); //CE : possible loss of precision**

**7. When ever we are assigning higher datatype value to lower datatype value variable by explicit type-casting ,the most significant bits will be lost i.e., we have considered least significant bits.**

**Example 3 :**

**int x=150;**

**short s=(short)x;**

**byte b=(byte)x;**

**System.out.println(s); //150**

**System.out.println(b); //-106**

**8. When ever we are assigning floating point value to the integral types by explicit type casting , the digits of after decimal point will be lost .**

**Example 4:**

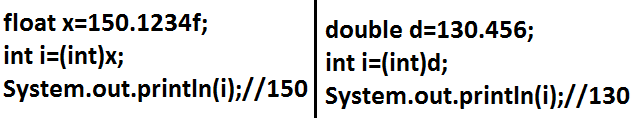
**double d=130.456 ;**

**int x=(int)d ;**

**System.out.println(x); //130**

**byte b=(byte)d ;**

**System.out.println(b); //-206**

****

**Example program for explicit type casting**

**-----------------------------------------------------**

**public class TemperatureConverter {**

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

**// Example of explicit type casting**

**double fahrenheit = 77.0; // Temperature in Fahrenheit**

**// Formula to convert Fahrenheit to Celsius**

**int celsius = (int) ((fahrenheit - 32) \* 5 / 9);**

**// Print the result**

**System.*out*.println("Temperature in Fahrenheit: " + fahrenheit);**

**System.*out*.println("Temperature in Celsius (explicit): " + celsius);**

**}**

**}**

**Instance variables:**

* **If the value of a variable is varied from object to object such type of variables are called instance variables.**
* **For every object a separate copy of instance variables will be created.**
* **Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.**
* **Instance variables will be stored on the heap as the part of object.**
* **Instance variables should be declared with in the class directly but outside of any method or block or constructor.**
* **Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.**
* **But by using object reference we can access instance variables from static area.**

**Example:**

**public class StudentDetails{**

**String name=”arjuin”;// instance variables**

**int rollNo=102;// instance variables**

**int marks=50;// instance variable.**

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

**// object creation**

**StudentDetails std=new StudentDetails();**

**System.out.println("Student Name : "+std.name);**

**System.out.println("Student RollNumber : "+std.rollNo);**

**System.out.println("Student Marks : "+std.marks);**

**}**

**}**

**For the instance variables it is not required to perform initialization JVM will always provide default values.  
  
 Example:**

**public class StudentDetails{**

**String name;// instance variables**

**int rollNo;// instance variables**

**int marks;// instance variable.**

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

**// object creation**

**StudentDetails std=new StudentDetails();**

**System.out.println("Student Name : "+std.name);**

**System.out.println("Student RollNumber : "+std.rollNo);**

**System.out.println("Student Marks : "+std.marks);**

**}**

**}**

**Output:**

**—-------**

**Null**

**0**

**0**

**Static variables:**

* **If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.**
* **In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.**
* **Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.**
* **Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor.**
* **Static variables can be accessed from both instance and static areas directly.**
* **We can access static variables either by class name or by object reference but usage of class name is recommended.**
* **But within the same class it is not required to use class name we can access directly.**

**Example**

**—----------**

**public class StudentDetails{**

**static String studentName="malli";// static variables**

**static int rollNumber=101;// static variables**

**static int studentMarks=98;// static variable.**

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

**// object creation**

**StudentDetails std=new StudentDetails();**

**System.out.println("Student Name : "+studentName);**

**System.out.println("Student RollNumber : "+std.rollNumber);**

**System.out.println("Student Marks : "+StudentDetails.studentMarks);**

**}**

**}**

**For the static variables it is not required to perform initialization explicitly, JVM will always provide default values.  
  
 Example:**

**public class StudentDetails{**

**static String studentName;// static variables**

**static int rollNumber;// static variables**

**static int studentMarks;// static variable.**

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

**// object creation**

**StudentDetails std=new StudentDetails();**

**System.out.println("Student Name : "+studentName);**

**System.out.println("Student RollNumber : "+std.rollNumber);**

**System.out.println("Student Marks : "+StudentDetails.studentMarks);**

**}**

**}**

**Output:**

**—-------**

**Null**

**0**

**0**

**Local variables:**

**Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.**

**Local variables will be stored inside stack.**

**The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.**

**public class LocalVariableExample {**

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

**// Local variables**

**int number1 = 10; // Declaring and initializing the first local variable**

**int number2 = 20; // Declaring and initializing the second local variable**

**// Calculate the sum of the two local variables**

**int sum = number1 + number2; // Declaring and initializing a third local variable**

**// Display the result**

**System.out.println("The sum of " + number1 + " and " + number2 + " is " + sum);**

**}**

**}**

**Console(I/o)**

**Console (i/o)**

**—--------------------**

**Input: program obtain values to variable during the runtime from user**

**1. Import java.util.Scanner;**

**2. Scanner sc=new Scanner(System.in);**

**Scanner class**

**Scanner class is present in java.util package. This class is useful to accept different types of input in the same line.**

**1.create scanner object connecting to the keyboard.**

**Syntax:**

**Scanner sc=new Scanner(System.in);**

**2.now all the input values are stored in the scanner object. Retrieve those values using the following methods.**

**(a)sc.next();àto read words**

**(b)sc.nextLine();àto read strings**

**(c)sc.nextInt();àto read integers**

**(d)sc.nextFloat();àto read float values**

**(e)sc.nextByte();àto read byte values**

**(f)sc.nextDouble();àto read double values**

**(g)sc.next().charAt(0);àto read characters**

**Write a program to obtain two int inputs from user find the sum**

**/sum of 5 and 8 is 13**

**import java.util.Scanner;**

**public class SumCalculator {**

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

**// Creating a Scanner object to read user input**

**Scanner scanner = new Scanner(System.*in*);**

**// Prompting the user to enter the first integer**

**System.*out*.print("Enter the first integer: ");**

**int num1 = scanner.nextInt(); // Reading the first integer input**

**// Prompting the user to enter the second integer**

**System.*out*.print("Enter the second integer: ");**

**int num2 = scanner.nextInt(); // Reading the second integer input**

**// Calculating the sum of the two integers**

**int result = num1 + num2;**

**// Displaying the result**

**System.*out*.println("Sum of " + num1 + " and " + num2 + " is " + result);**

**// Closing the scanner**

**scanner.close();**

**}**

**}**

**Calculate Area of triangle using it base and height**

**import java.util.Scanner;**

**import java.util.Scanner;**

**public class TriangleAreaCalculator {**

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

**// Create a Scanner object to read input from the user**

**Scanner scanner = new Scanner(System.*in*);**

**// Prompt the user to enter the base of the triangle**

**System.*out*.print("Enter the base of the triangle (in meters): ");**

**double base = scanner.nextDouble(); // Read the base input**

**// Prompt the user to enter the height of the triangle**

**System.*out*.print("Enter the height of the triangle (in meters): ");**

**double height = scanner.nextDouble(); // Read the height input**

**// Calculate the area of the triangle**

**double area = 0.5 \* base \* height;**

**// Display the calculated area**

**System.*out*.println("Area of the triangle with base " + base + " meters and height " + height + " meters is: "**

**+ area + " square meters");**

**// Close the Scanner object to release resources**

**scanner.close();**

**}**

**}**

**Limitations of Console(i/o)**

**-----------------------------**

**Console input and output (I/O) in Java refers to interactions between a Java program and the console or terminal where it runs. While console I/O is straightforward and useful for simple applications and debugging, it has some limitations compared to more advanced I/O techniques. Here are the main limitations of console I/O statements in Java:**

**1. Limited Interaction: Console I/O is primarily text-based and does not support complex interactions or graphical user interfaces (GUIs). It's suitable for basic input and output operations but lacks the ability to handle more advanced user interactions or visual elements.**

**2. No Formatting Options: Console output is typically plain text with limited formatting options. While basic formatting (like newline characters or tabs) is possible, advanced formatting (like colors, fonts, or alignment) is not supported directly through console I/O statements.**

**3. No Input Validation: Console input lacks built-in mechanisms for input validation. Developers must manually validate and handle user inputs to ensure they meet specific criteria (e.g., checking for valid data types or ranges), which can lead to more error-prone code.**

**4. Single Threaded: Console I/O operations in Java are typically synchronous and block the execution of the program until the operation completes. This can lead to performance issues in applications requiring concurrent input/output or responsiveness.**

**5. Platform Dependency: While Java itself is platform-independent, console I/O can behave differently across different operating systems. For instance, handling special characters or controlling terminal behavior might vary between Windows, macOS, and Linux systems.**

**6. Limited User Experience: For end-users, console-based applications may provide a less engaging user experience compared to applications with graphical interfaces. Visual feedback, interactive controls, and multimedia elements are not supported through console I/O.**

**7. No Localization Support: Console I/O does not inherently support internationalization and localization features, such as displaying messages in different languages or handling locale-specific input formats (e.g., dates or numeric separators).**

**Operators and Operands**

**An Operator is a symbol that represents a specific operation on one or more operands.**

**Eg. a+b;**

**In the above, a and b are called operands**

**And ‘+’ is called an operator.**

**Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:**

**· Arithmetic Operators**

**· Relational Operators**

**· Bitwise Operators**

**· Logical Operators**

**· Assignment Operators**

**· Unary Operator**

**· Ternary Operator**

**· Shift Operators**

**Arithmetic Operators**

**---------------------------------**

**Arithmetic operators example program**

**—------------------------------------**

**package com.codegnan.operators;**

**public class Main {**

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

**// Example 1: Finance and Business**

**double revenue = 15000.0;**

**double expenses = 10000.0;**

**double profit = revenue - expenses;**

**double profitMargin = (profit / revenue) \* 100;**

**System.*out*.println("Profit Margin: " + profitMargin + "%");**

**// Example 2: Physics and Engineering (Projectile Motion)**

**double initialVelocity = 30.0; // m/s**

**double launchAngle = 45.0; // degrees**

**double horizontalVelocity = initialVelocity \* Math.*cos*(Math.*toRadians*(launchAngle));**

**double verticalVelocity = initialVelocity \* Math.*sin*(Math.*toRadians*(launchAngle));**

**System.*out*.println("Horizontal Velocity: " + horizontalVelocity + " m/s");**

**System.*out*.println("Vertical Velocity: " + verticalVelocity + " m/s");**

**// Example 3: Gaming and Simulation (Player Health)**

**int initialHealth = 100;**

**int damage = 20;**

**int remainingHealth = initialHealth - damage;**

**System.*out*.println("Remaining Health: " + remainingHealth);**

**// Example 4: Education and Training (Grading System)**

**int exam1Score = 85;**

**int exam2Score = 78;**

**int exam3Score = 92;**

**double averageScore = (exam1Score + exam2Score + exam3Score) / 3.0;**

**System.*out*.println("Average Score: " + averageScore);**

**// Example 5: IoT and Sensor Data Processing (Temperature Conversion)**

**double celsiusTemperature = 25.0;**

**double fahrenheitTemperature = (celsiusTemperature \* 9 / 5) + 32;**

**System.*out*.println("Fahrenheit Temperature: " + fahrenheitTemperature + " °F");**

**}**

**}**

## **String Concatenation operator :**

**1. The only overloaded operator in java is ' + ' operator some times it access arithmetic addition operator & some times it access String concatenation operator.**

**2. If acts as one argument is String type , then '+' operator acts as concatenation and If both arguments are number type , then operator acts as arithmetic operator**

**public class Example {**

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

**// Initialize variables**

**String text = "ashok";**

**int num1 = 10;**

**int num2 = 20;**

**int num3 = 30;**

**// Output concatenation of string and integers**

**System.out.println(text + num1 + num2 + num3); // Output: ashok102030**

**// Output sum of integers followed by string**

**System.out.println(num1 + num2 + num3 + text); // Output: 60ashok**

**// Output sum of two integers, string, and another integer**

**System.out.println(num1 + num2 + text + num3); // Output: 30ashok30**

**// Output integer, string, and concatenation of remaining integers**

**System.out.println(num1 + text + num2 + num3); // Output: 10ashok2030**

**}**

**}**

### Comparison Operators in Java or relational operators

To compare two values (or variables), comparison operators are used. This is crucial to programming since it facilitates decision-making and the search for solutions. A comparison's return value is either true or false. These are referred to as "Boolean values."

| **Operators** | **Operations** |
| --- | --- |
| == | Equal to |
| != | Not equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

#### Example

**public** **class** ComparisonOperatorsExample {

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

**int** num1 = 10;

**int** num2 = 20;

// Equal to (==) operator

**boolean** isEqual = (num1 == num2);

System.***out***.println("num1 is equal to num2: " + isEqual);

// Not equal to (!=) operator

**boolean** isNotEqual = (num1 != num2);

System.***out***.println("num1 is not equal to num2: " + isNotEqual);

// Greater than (>) operator

**boolean** isGreaterThan = (num1 > num2);

System.***out***.println("num1 is greater than num2: " + isGreaterThan);

// Less than (<) operator

**boolean** isLessThan = (num1 < num2);

System.***out***.println("num1 is less than num2: " + isLessThan);

// Greater than or equal to (>=) operator

**boolean** isGreaterThanOrEqual = (num1 >= num2);

System.***out***.println("num1 is greater than or equal to num2: " + isGreaterThanOrEqual);

// Less than or equal to (<=) operator

**boolean** isLessThanOrEqual = (num1 <= num2);

System.***out***.println("num1 is less than or equal to num2: " + isLessThanOrEqual);

}

}

### Logical Operators in Java

Logical Operators in Java check whether the expression is true or false. It is generally used for making any decisions in Java programming. Not only that but [Jump statements in Java](https://www.scholarhat.com/tutorial/java/java-jump-statements-break-continue-return) are also used for checking whether the expression is true or false. It is generally used for making any decisions in Java programming.

| **Operators** | **Example** | **Meaning** |
| --- | --- | --- |
| && [ logical AND ] | expression1 && expression2 | (true) only if both of the expressions are true |
| || [ logical OR ] | expression1 || expression2 | (true) if one of the expressions in true |
| ! [ logical NOT ] | !expression | (true) if the expression is false and vice-versa |

#### Logical Operators in Java Example

**class** Main {

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

// && operator

System.***out***.println((6 > 3) && (8 > 6)); // true

System.***out***.println((6 > 3) && (8 < 6)); // false

// || operator

System.***out***.println((6 < 3) || (8 > 6)); // true

System.***out***.println((6 > 3) || (8 < 6)); // true

System.***out***.println((6 < 3) || (8 < 6)); // false

// ! operator

System.***out***.println(!(6 == 3)); // true

System.***out***.println(!(6 > 3)); // false

}

}

### Unary Operator in Java

Unary Operators in Java are used in only one operand. There are various types of Unary Operators in Java, such as

| **Operators** | **Description** |
| --- | --- |
| + | Unary Plus |
| - | Unary Minus |
| ++ | Increment operator |
| -- | Decrement Operator |
| ! | Logical complement operator |

**class** Main

{

**public** **static** **void** main(String[] args)

{

// declare variables

**int** a = 13, b = 13;

**int** result1, result2;

// original value

System.***out***.println("Value of a: " + a);

// increment operator

result1 = ++a;

System.***out***.println("After increment: " + result1);

System.***out***.println("Value of b: " + b);

// decrement operator

result2 = --b;

System.***out***.println("After decrement: " + result2);

}

}

### Bitwise Operators in Java

Bitwise Operators in Java are used to assist the performance of the operations on individual bits. There are various types of Bitwise Operators in Java, such as. We will see the working of the Bitwise Operators in the [**Java Online Compiler**](https://www.scholarhat.com/compiler/java).

| **Operators** | **Descriptions** |
| --- | --- |
| ~ | Bitwise Complement |
| << | Left shift |
| >> | Right shift |
| >>> | Unsigned Right shift |
| & | Bitwise AND |
| ^ | Bitwise exclusive OR |

#### Bitwise Operators in Java Example

**package** com.codegnan.operators;

**import** java.util.Scanner;

**public** **class** BitwiseOperatorsExample {

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

Scanner scanner = **new** Scanner(System.***in***);

// Prompt the user to enter two integers

System.***out***.print("Enter integer 1: ");

**int** num1 = scanner.nextInt();

System.***out***.print("Enter integer 2: ");

**int** num2 = scanner.nextInt();

// Perform bitwise operations

**int** bitwiseAnd = num1 & num2;

**int** bitwiseOr = num1 | num2;

**int** bitwiseXor = num1 ^ num2;

**int** bitwiseComplementNum1 = ~num1;

**int** leftShiftNum1 = num1 << 1;

**int** rightShiftNum2 = num2 >> 1;

**int** unsignedRightShiftNum2 = num2 >>> 1;

// Display results of bitwise operations

System.***out***.println("Bitwise AND (" + num1 + " & " + num2 + ") : " + bitwiseAnd);

System.***out***.println("Bitwise OR (" + num1 + " | " + num2 + ") : " + bitwiseOr);

System.***out***.println("Bitwise XOR (" + num1 + " ^ " + num2 + ") : " + bitwiseXor);

System.***out***.println("Bitwise NOT (~" + num1 + ") : " + bitwiseComplementNum1);

System.***out***.println("Left Shift (" + num1 + " << 1) : " + leftShiftNum1);

System.***out***.println("Right Shift (" + num2 + " >> 1) : " + rightShiftNum2);

System.***out***.println("Unsigned Right Shift (" + num2 + " >>> 1) : " + unsignedRightShiftNum2);

// Close the scanner

scanner.close();

}

}

### Comparison Operators in Java or relational operators

To compare two values (or variables), comparison operators are used. This is crucial to programming since it facilitates decision-making and the search for solutions. A comparison's return value is either true or false. These are referred to as "Boolean values."

| **Operators** | **Operations** |
| --- | --- |
| == | Equal to |
| != | Not equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

#### Example

**public** **class** ComparisonOperatorsExample {

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

**int** num1 = 10;

**int** num2 = 20;

// Equal to (==) operator

**boolean** isEqual = (num1 == num2);

System.***out***.println("num1 is equal to num2: " + isEqual);

// Not equal to (!=) operator

**boolean** isNotEqual = (num1 != num2);

System.***out***.println("num1 is not equal to num2: " + isNotEqual);

// Greater than (>) operator

**boolean** isGreaterThan = (num1 > num2);

System.***out***.println("num1 is greater than num2: " + isGreaterThan);

// Less than (<) operator

**boolean** isLessThan = (num1 < num2);

System.***out***.println("num1 is less than num2: " + isLessThan);

// Greater than or equal to (>=) operator

**boolean** isGreaterThanOrEqual = (num1 >= num2);

System.***out***.println("num1 is greater than or equal to num2: " + isGreaterThanOrEqual);

// Less than or equal to (<=) operator

**boolean** isLessThanOrEqual = (num1 <= num2);

System.***out***.println("num1 is less than or equal to num2: " + isLessThanOrEqual);

}

}

### Ternary Operators in Java

The only conditional operator that accepts three operands is the ternary operator in Java. Java programmers frequently use it as a one-line alternative to the if-then-else expression. The ternary operator can be used in place of if-else statements, and it can even be used to create switch statements with nested ternary operators. The conditional operator uses less space and aids in writing if-else statements as quickly as possible even if it adheres to the same algorithm as an if-else statement

### **Syntax of Ternary Operator in java**

****Syntax: (condition)? expression1 : expression2;

Ternary operators example

—---------------------------------

**package** com.codegnan.operators;

**import** java.util.Scanner;

**public** **class** EligibilityCheck {

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

// Create a Scanner object for input

Scanner scanner = **new** Scanner(System.***in***);

// Prompt the user to enter their age

System.***out***.print("Enter your age: ");

**int** age = scanner.nextInt();

// Use ternary operator to check eligibility to vote

String eligibility = (age >= 18) ? "You are eligible to vote!" : "You are not eligible to vote yet.";

// Output the eligibility result

System.***out***.println(eligibility);

// Close the scanner

scanner.close();

}

}

Nested ternary operator example

—----------------------------------------

Ex 1 :

int x=(10>20)?30:((40>50)?60:70);

System.out.println(x); //70

**package** com.codegnan.operators;

**import** java.util.Scanner;

**public** **class** TernaryOperatorExample {

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

// Create a Scanner object for input

Scanner scanner = **new** Scanner(System.***in***);

// Prompt the user to enter two numbers

System.***out***.print("Enter two numbers, separated by space: ");

**int** num1 = scanner.nextInt();

**int** num2 = scanner.nextInt();

// Use nested ternary operators to determine the value of x

**int** x = (num1 > num2) ? 30 : ((num1 + num2 > 100) ? 60 : 70);

// Output the value of x

System.***out***.println("Value of x based on input: " + x);

// Close the scanner

scanner.close();

}

}

Example

----------------

**package** com.codegnan.controlstatements;

**import** java.util.Scanner;

**public** **class** CalculateRemainingAmount {

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

Scanner scanner = **new** Scanner(System.***in***);

// Prompt the user to enter the purchase amount

System.***out***.print("Enter the purchase amount in ₹: ");

**double** purchaseAmount = scanner.nextDouble();

// Applying discount based on purchase amount using conditional operators

**double** discount = (purchaseAmount >= 100) ? 0.2 : (purchaseAmount >= 50) ? 0.1 : 0.0;

// Calculating remaining amount after applying discount

**double** discountAmount = purchaseAmount \* discount;

**double** remainingAmount = purchaseAmount - discountAmount;

// Calculate percentage discount

**double** percentDiscount = discount \* 100;

// Output the remaining amount and percentage discount

System.***out***.println("Amount after applying " + percentDiscount + "% discount: ₹" + remainingAmount);

System.***out***.println("Discount applied: ₹" + discountAmount);

scanner.close();

}

}

Example-4

**import** java.util.Scanner;

/\*public class SecondGreatestNumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter three numbers

System.out.print("Enter the first number: ");

int num1 = scanner.nextInt();

System.out.print("Enter the second number: ");

int num2 = scanner.nextInt();

System.out.print("Enter the third number: ");

int num3 = scanner.nextInt();

// Calculate the second largest number

int secondLargest;

if (num1 >= num2 && num1 >= num3) {

// num1 is the largest

secondLargest = (num2 >= num3) ? num2 : num3;

} else if (num2 >= num1 && num2 >= num3) {

// num2 is the largest

secondLargest = (num1 >= num3) ? num1 : num3;

} else {

// num3 is the largest

secondLargest = (num1 >= num2) ? num1 : num2;

}

// Output the second largest number

System.out.println("The second largest number is: " + secondLargest);

scanner.close();

}

}\*/

Example-5

------------------

**public** **class** StudentGrade {

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

Scanner scanner = **new** Scanner(System.***in***);

// Input marks from the user

System.***out***.print("Enter student marks: ");

**int** marks = scanner.nextInt();

// Determine grade based on marks using ternary operator

**char** grade = (marks >= 90) ? 'A' :

(marks >= 80) ? 'B' :

(marks >= 70) ? 'C' :

(marks >= 60) ? 'D' :

(marks >= 50) ? 'E' : 'F';

// Output the grade

System.***out***.println("Student Grade: " + grade);

scanner.close();

}

}

Example-6

**public** **class** LeapYearChecker {

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

Scanner scanner = **new** Scanner(System.in);

System.out.print("Enter a year: ");

**int** year = scanner.nextInt();

**boolean** isLeapYear = (year % 4 == 0) && (year % 100 != 0 || year % 400 == 0);

String result = isLeapYear ? " is a leap year." : " is not a leap year.";

System.out.println(year + result);

scanner.close();

}

}

### Assignment Operator in Java

Assignment Operators are mainly used to assign the values to the variable that is situated in Java programming. There are various assignment operators in Java, such as

| **Operators** | **Examples** | **Equivalent to** |
| --- | --- | --- |
| = | X = Y; | X = Y; |
| += | X += Y; | X = X + Y; |
| -= | X -= Y; | X = X - Y; |
| \*= | X \*= Y; | X = X \* Y; |
| /= | X /= Y; | X = X / Y; |
| %= | X %= Y; | X = X % Y; |

#### Assignment Operator in Java Example

**class** Main {

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

// create variables

**int** a = 5;

**int** var;

// assign value using =

var = a;

System.***out***.println("var using =: " + var);

// assign value using =+

var += a;

System.***out***.println("var using +=: " + var);

// assign value using =\*

var \*= a;

System.***out***.println("var using \*=: " + var);

}

}