**Core Java Notes Topic Wise**

[Java Features 2](#_Toc188024651)

[Java History 2](#_Toc188024652)

[Structure of a Java Program 5](#_Toc188024653)

# Java Features

Java is one of the most widely used programming languages due to its versatility, portability, and performance. It has a rich set of features that make it suitable for developing a wide range of applications, from desktop applications to large-scale enterprise systems and mobile apps. Here are the key features of the Java programming language:

**1. Simple:**

Clear Syntax: Java’s syntax is similar to C++, but it eliminates the complexity found in C++. For example, Java does not have pointers, which reduces the complexity of memory management and eliminates a class of bugs.

No Complex Features: Features like operator overloading and multiple inheritance (of classes) are not present in Java, which makes it simpler and easier to learn.

**2. Object-Oriented**

Everything is an Object: In Java, everything is treated as an object (except for primitive types). This includes encapsulation, inheritance, and polymorphism as core principles.

Classes and Objects: Java encourages organizing software around objects and classes, which allows for better modularity, reusability, and maintainability.

Abstraction: Java allows abstraction through interfaces and abstract classes, helping hide the internal implementation details from the user.

**3. Platform-Independent (Write Once, Run Anywhere)**

Java Bytecode: Java code is compiled into bytecode, which is platform-independent. The bytecode can be run on any machine that has a Java Virtual Machine (JVM), making Java a highly portable language.

JVM (Java Virtual Machine): The JVM abstracts the underlying operating system and hardware, enabling Java programs to run consistently on any platform without modification.

Cross-Platform Development: Java’s portability is a significant advantage for developers, as they can write code once and run it anywhere, be it on Windows, Linux, or macOS.

**4. Robust**

Automatic Memory Management: Java has automatic garbage collection, which helps in reclaiming memory by automatically removing objects that are no longer in use, thus preventing memory leaks.

Exception Handling: Java provides robust exception handling mechanisms using try, catch, throw, and throws keywords, which help in dealing with runtime errors effectively.

Type Checking: Java is strongly typed, which ensures that variables and objects are checked at compile time. This helps in catching errors early in the development process.

Null Pointer Handling: Java provides built-in support for handling null references in a controlled manner, preventing common errors such as null pointer exceptions.

***5. Multithreaded***

Built-In Support for Multithreading: Java provides built-in support for multithreading, allowing developers to create highly efficient and concurrent applications. With the Thread class and Runnable interface, Java makes it easy to perform multiple tasks simultaneously.

Synchronization: Java includes synchronization mechanisms to manage access to shared resources in multi-threaded environments, ensuring thread safety.

Concurrency Utilities: Java's java.util.concurrent package includes several utilities that simplify the development of multi-threaded applications, such as thread pools, executors, and locks.

***6. Distributed Computing***

Networking Support: Java provides a rich API for developing networked applications, including classes for implementing TCP/IP and UDP protocols. The java.net package allows developers to create both client and server-side applications with ease.

Remote Method Invocation (RMI): RMI enables Java objects to communicate with each other over a network, making it easier to develop distributed applications.

Java Beans and EJB: Java supports enterprise-level distributed applications through JavaBeans and Enterprise JavaBeans (EJB), which allow the development of reusable and modular components.

***7. High Performance***

Just-in-Time Compilation (JIT): Java code is first compiled into bytecode, and then the JVM uses a Just-in-Time compiler to convert bytecode into machine code at runtime. This improves performance, as code is optimized for the underlying hardware.

Optimized Garbage Collection: Java’s garbage collection process is highly optimized, reducing the performance overhead of manual memory management found in other languages.

***8. Secure***

Security Manager: Java’s security model is designed to protect against unauthorized access to system resources. The Security Manager and Bytecode Verifier ensure that Java programs do not perform unsafe operations.

Bytecode Verification: Before being executed by the JVM, bytecode is verified to ensure it does not contain illegal or harmful operations, providing a layer of security for Java applications, especially in web and network environments.

Access Control: Java offers strong access control mechanisms (e.g., public, private, protected, and package-private) to restrict unauthorized access to fields and methods.

***9. Dynamic***

Dynamic Loading: Java allows the dynamic loading of classes at runtime using reflection. This capability makes Java suitable for creating flexible, extensible systems that can change at runtime without needing to be recompiled.

Reflection API: The Reflection API allows Java programs to inspect and manipulate objects and classes during runtime, enabling dynamic behavior and increasing flexibility.

***10. Rich Standard Library (API)***

Java provides an extensive set of libraries and APIs that cover everything from file I/O, networking, and database connectivity (JDBC), to graphical user interface (GUI) development (Swing, JavaFX), and XML parsing.

Standard Libraries: The Java standard library includes packages such as java.util for collections and utilities, java.io for file handling, java.math for math operations, java.time for date and time handling, and more.

***11. Portable***

Java programs can be run on any device that has the JVM installed, whether it's a desktop computer, a server, or a mobile device. The JVM abstracts the underlying operating system and hardware, ensuring portability.

Java for Mobile Development: With technologies like Java ME (Micro Edition) and Android (which relies on Java-based frameworks), Java is widely used for developing applications across a broad range of devices.

***12. Scalable***

Java is used in large-scale enterprise applications, and its design allows for scalability. It provides tools such as load balancing, threading, and databases (via JDBC) to build scalable solutions.

Enterprise Features: Java Enterprise Edition (Java EE) offers features like Enterprise JavaBeans (EJB), Java Message Service (JMS), and Java Database Connectivity (JDBC) for enterprise-level applications, allowing systems to scale efficiently.

***13. Community and Ecosystem***

Open Source and Rich Ecosystem: Java has a vast ecosystem of open-source libraries, frameworks, and tools, such as Spring, Hibernate, and Apache libraries, which make development faster and easier.

Active Community: Java has a large, active developer community, providing extensive documentation, forums, conferences, and support to Java developers worldwide.

# Java History

***Java 5 (2004):***

Generics: Introduced type safety and improved code readability.

Autoboxing/Unboxing: Simplified conversions between primitive types and their corresponding wrapper classes.

Enhanced for loop: A concise way to iterate over arrays and collections.

Enums: Provided a type-safe way to define a set of constants.

VarArgs: Allowed methods to accept a variable number of arguments.

***Java 6 (2006):***

Scripting support: Integrated scripting languages like JavaScript and Python.

JDBC 4.0: Enhanced database connectivity with improved performance and features.

Pluggable annotations: Allowed custom annotations to be processed at runtime.

Performance improvements: Optimized the JVM for better performance.

***Java 7 (2011):***

Diamond operator: Simplified generic instantiation.

Try-with-resources: Automatically closed resources at the end of a try block.

String in switch: Allowed strings to be used as switch case labels.

Binary literals: Introduced binary literals using the 0b prefix.

Numeric literal underscores: Improved readability of large numbers.

***Java 8 (2014):***

Lambda expressions: Functional programming style for concise code.

Streams API: A powerful way to process collections of data.

Default and static methods in interfaces: Enhanced interface capabilities.

Date and Time API: A more flexible and robust API for date and time operations.

Optional class: Handled optional values gracefully.

***Java 9 (2017):***

Module system (Project Jigsaw): Improved modularity and encapsulation.

JShell: A REPL for interactive Java programming.

HTTP/2 client: A modern HTTP client with improved performance and features.

Reactive Streams API: Support for asynchronous and non-blocking programming.

***Java 10 (2018):***

Local variable type inference: Declared variables without explicitly specifying their type.

Parallel full GC for G1: Improved garbage collection performance.

***Java 11 (2018):***

LTS release: Long-term support for production environments.

HTTP client API: A standardized HTTP client API.

Lambda parameters: Local-variable syntax for lambda parameters.

String methods: New string methods like isBlank(), strip(), and stripLeading().

***Java 12 (2019):***

Switch expressions: Enhanced switch statements with pattern matching.

File system API: A modern file system API for better file I/O operations.

***Java 13 (2019):***

Text blocks: Multi-line strings with automatic formatting.

Switch expressions (preview): Further improvements to switch expressions.

***Java 14 (2020):***

Pattern matching for instanceof: More concise pattern matching for instanceof checks.

Records: A concise way to define immutable data classes.

***Java 15 (2020):***

Sealed classes: Restricted class inheritance.

Hidden classes: Improved class file format to support dynamic language features.

***Java 16 (2021):***

Records (second preview): Further refinements to records.

Pattern matching for instanceof (second preview): Additional features for pattern matching.

***Java 17 (2021):***

LTS release: Long-term support for production environments.

Sealed classes (second preview): Further refinements to sealed classes.

Pattern matching for instanceof (standard): Standardized pattern matching for instanceof.

***Java 18 (2022):***

UTF-8 by default: UTF-8 as the default charset.

Simple Web Server: A simple web server for testing purposes.

Code snippets in Java API documentation: Improved documentation with code examples.

***Java 19 (2023):***

Record patterns: Pattern matching for records.

Foreign Function & Memory API (Third Incubator): Interoperability with native code.

Vector API (Fourth Incubator): Improved vector operations.

***Java 20 (2023):***

Record patterns (Second Preview): Further refinements to record patterns.

Pattern Matching for switch (Third Preview): Additional features for pattern matching in switch statements.

***Java 21 (2023):***

Record patterns (Standard): Standardized record patterns.

Pattern Matching for switch (Standard): Standardized pattern matching for switch statements.

Virtual Threads: Lightweight user-mode threads.

***Java 22 (2023):***

Scoped Values: Local variables with limited scope.

Pattern Matching for switch (Preview): Additional features for pattern matching in switch statements.

# Structure of a Java Program

A Java program is a collection of one or more classes. Each class contains methods & variables that define the behavior & state of objects of that class.

***Basic Structure:***

public class MyClass {

public static void main(String[] a)

{

// Main execution block

System.out.println("Hello, Welcome to the Java Classes ....!");

}

}

***1. Class Declaration:***

- public class MyClass:

- public: Access modifier, making the class accessible from anywhere.

- class: Keyword to define a class.

- MyClass: Name of the class.

***2. Main Method:***

- public static void main(String[] args):

- public: Access modifier, making the method accessible from anywhere.

- static: Keyword indicating that the method belongs to the class, not to an instance of the class.

- void: Return type, indicating that the method doesn't return any value.

- main: Special method name, the entry point of a Java application.

- String[] args: An array of strings, used to pass command-line arguments to the program.

***QA:***

1.Why the method name is only main.?

Ans:Sunmicro team had a problem of where to start the execution...?

developer1 said-> start from test method

developer2 said -> start from demo method

developer3 said-> start from example method

* if a application is developed by developer1 and he said start from the test and later is resigned and left, later developer2 is joined he has to find the method from in which file the starting method is there and name of the method.
* to over come this problem sun team they said hey look developers, to overcome this problem we are fixing the name of the method name to main so every java programmer who wants to write java code has to start from main in this way the compiler is designed.

2.is it compulsory to write the method name as main..?

yes to start the program execution it is compulsory but for compiling the code it is not required.

3.what will happen if i write other name to a method..?

nothing will happen it will just act like a normal method

4. can we overload the main method..?

we will discuss this in the methods

5. can a class can have multiple main methods ..?

yes but it should differ in parameters otherwise compile time error. in detail we will discuss in the methods concept

***3. Method Body:***

- The code within the curly braces {} is the method's body.

- System.out.println("Hello, World!"); : This line prints the message "Hello, World!" to the console.

Example: Addition of Two Numbers:

public class Sum{

public static void main(String[] args) {

int num1 = 10;

int num2 = 5;

int sum = num1 + num2;

System.out.println("The sum of two numbers is:" + sum); } }

***Key Points:***

- Class: A blueprint for creating objects.

- Object: An instance of a class.

- Method: A block of code that performs a specific task.

- Variable: A container for storing data.

- Data Types: Java has various data types, including int, double, String, Boolean, etc.

- Operators: Java supports arithmetic, comparison, logical, and bitwise operators.

- Control Flow Statements: if-else, switch, for, while, and do-while statements control the flow of execution.

***Task:***

1.Re write the above two programs and execute.?

# Compilation and Execution of Java Programs:

***1.Compilation:***

Source Code: You write Java code in a text editor and save it with a .java extension.

Compiler: The Java compiler, javac, reads the source code and checks for syntax errors and other potential issues.

***Bytecode Generation:***

If there are no errors, the compiler translates the source code into bytecode,

which is platform-independent. This bytecode is stored in a .class file.

***2.Execution:***

JVM: The Java Virtual Machine (JVM) is a software layer that interprets and executes the bytecode.

Class Loading: The JVM loads the necessary class files into memory.

Bytecode Verification: The JVM verifies the bytecode to ensure it follows Java's security and integrity rules.

Bytecode Interpretation: The JVM interprets the bytecode and executes it, line by line.

Just-In-Time (JIT) Compilation:

For performance optimization, The JVM can compile frequently executed parts of the bytecode into native machine code. This speeds up execution.

***Example:***

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

public class Sub{

public static void main(String[] args) {

int i = 20;

int j = 30;

System.out.println("The Subtraction of the " + i + ", " + j + " is : " + (i-j));

}

}

***Compilation:***

**javac Sub.java**

this command will genarate the .class file which jvm will use at the time of the execution.

***Execution:***

**java Sub**

this command will starts the jvm and loads the .class file and executes the main method.

***Output:***

The Subtraction of the 20, 30 is : -10

***Task:***

1.Write any two java programs, compile and run the program observe the output …?

# Reserved Words:

Reserved words in java have some predefined meaning to it so we can't use those words for the other purpose like identifiers.

Refer: ReservedWords\_Chart.png

***Literal***

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

Literal is a constant value which we can assign to a variable is called literals

**Example:**

String name = "Rohit Sharma";

String is a datatype

name is a identifiers

"Rohit Sharma" is a literal.

-> All the reserved words in java starts with lowercase latters.

# Data Types:

Every variable or Expression has a type in java. and all the types are strictly typed in java.

because java is strictly typed language the compiler role is to check the literal stored in the variable can handle it or not

This checking is called type Checking / Strictly types checking.

Datatypes

Primitive Datatype

a. whole number

1. byte(1byte)

2. short(2bytes)

3. int(4bytes) \* commonly used datatype

4. long(8 bytes)

b. realnumber

1. float

2. double

Non-Primitive DataTypes:

String

Object

List

Tuple

Stack

Queue...... etc.

Primitive DataTypes:

====================

1.byte

2.short

3.int

4.long

5.float

6.double

7.boolean

Note : the information of the datatypes are given in the following way

1.size of the data types

2.min values it can store

3.max value it can store

byte:

=====

byte:

size -> 8 bits

minvalue -> -128

maxvalue -> 127

System.out.println("Byte Size :: "+ Byte.SIZE);

System.out.println("MINVALUE of byte is :: "+ Byte.MIN\_VALUE);

System.out.println("MAXVALUE of byte is :: "+ Byte.MAX\_VALUE);

short:

======

size : 16bits(2 byte)

minvalue: -32768

maxvalue: +32767

System.out.println("Size of int is :: "+Short.SIZE);

System.out.println("MINVALUE of int is :: "+Short.MIN\_VALUE);

System.out.println("MAXVALUE of intt is :: "+Short.MAX\_VALUE);

int:

====

size: 32bits(4 bytes)

minvalue:-2147483648

maxvalue: 2147483647

System.out.println("Size of int is :: "+Integer.SIZE);

System.out.println("MINVALUE of int is :: "+Integer.MIN\_VALUE);

System.out.println("MAXVALUE of intt is :: "+Integer.MAX\_VALUE);

Note:

=====

By default the compile will store the whole numbers as a int only. we can also store the values in the byte and short also.

long:

=====

size: 64bits(8bytes)

minvalue:-9223372036854775808

maxvalue:9223372036854775807

System.out.println("Size of long is :: "+Long.SIZE);

System.out.println("MINVALUE of long is :: "+Long.MIN\_VALUE);

System.out.println("MAXVALUE of long is :: "+Long.MAX\_VALUE);

note:

=====

if you want to store the values in the long type then you must suffix the value with either "L" or "l".

Otherwise compile time error

Ex:

long a = 123l;

long b = 123; //CE

float:

======

size: 32 bits(1 byte = 8bits, 32/8 = 4 bytes)

minvalue:1.4E-45

maxvalue: 3.4028235E38

System.out.println("Size of float is :: "+Float.SIZE);

System.out.println("MINVALUE of float is :: "+Float.MIN\_VALUE);

System.out.println("MAXVALUE of float is :: "+Float.MAX\_VALUE);

Note:

=====

By default compiler will store the realNumbers/ decimalNumbers as double values if you want to store it a float,

then you have to suffix with "f" or "F" other wise CE.

double:

=======

size: 64s bits(1 byte = 8bits, 64/8 = 8 bytes)

minvalue:4.9E-324

maxvalue: 1.7976931348623157E308

System.out.println("Size of double is :: "+ Double.SIZE);

System.out.println("MINVALUE of double is :: "+ Double.MIN\_VALUE);

System.out.println("MAXVALUE of double is :: "+ Double.MAX\_VALUE);

Note:

======

To map primitive data as Object in java from JDK1.5 concept of "Wrapper class" was introduced in JDK1.5 version.

respective Object classes of primitive datatypes.

byte -------------> Byte(C)

short--------------> Short(C)

int --------------> Integer(C)

long --------------> Long(C)

float --------------> Float(C)

double------------> Double(C)