Unit – 1
Java Overview

Introduction to Java

- B led to C, C evolved into C++, and C++ set the stage for Java.
- **James Gosling** (Father of Java) , Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at **Sun Microsystems**.
- It took 8 months to develop the first working version called “Oak”
- Renamed as "Java" in 1995.

Java language Specification, API

Java Language Specification:

The Java language specification is a technical definition of the Java programming language's syntax and semantics.

API: The application program interface (API) also known as library, contains predefined classes and interfaces for developing Java programs.

Java SE (Standard Edition) : for Client side programming.
Java EE (Enterprise Edition) : for Server side programming.
Java ME (Micro Edition) : for Mobile devices.

JDK, IDE, JVM

JDK:

Java Development Toolkit: The JDK consists of a set of separate programs, each invoked from a command line, for developing and testing Java programs.

IDE:

Integrated Development Environment: It is used for developing Java programs quickly where Editing, compiling, building, debugging, and online help are integrated in one graphical user interface.

Ex. NetBeans, Eclipse etc.

JDK, IDE, JVM

JVM:

JVM (Java Virtual Machine) is a software “engine” that runs Java programs (and other JVM languages like Kotlin/Scala).

Java code is first compiled into bytecode (.class files), not directly into machine code.

The same bytecode runs on any OS (Windows/Linux/macOS) as long as a JVM exists → “Write Once, Run Anywhere.”

JVM’s main job: load classes, verify bytecode, execute instructions, and manage memory.

Platform Independence

- Java is platform independent because it doesn’t compile directly to OS-specific machine code.
- Java source code (.java) is compiled by javac into bytecode (.class).
- Bytecode is the same for all platforms (Windows/Linux/macOS).
- Each platform has its own JVM, and the JVM converts bytecode into native machine code for that OS/CPU.
- So the same program can run anywhere with a JVM installed → “Write Once, Run Anywhere (WORA)”.

Bytecode

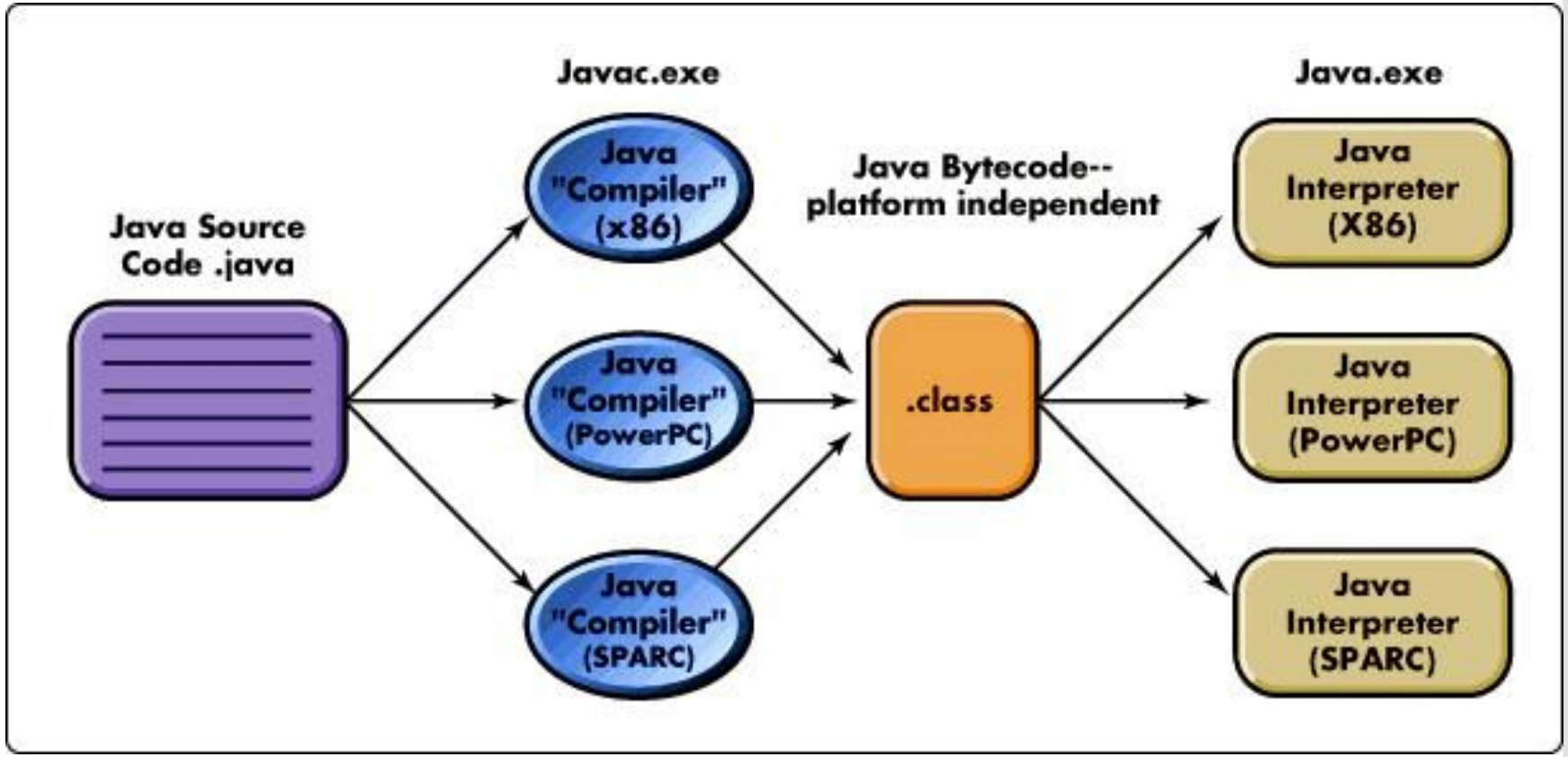
Java bytecode is the instruction set for the Java Virtual Machine.

As soon as a java program is compiled, java bytecode is generated in the form of a .class file

Highly optimized set of instructions designed to be executed by

the Java run-time system, which is called the Java Virtual Machine(JVM)

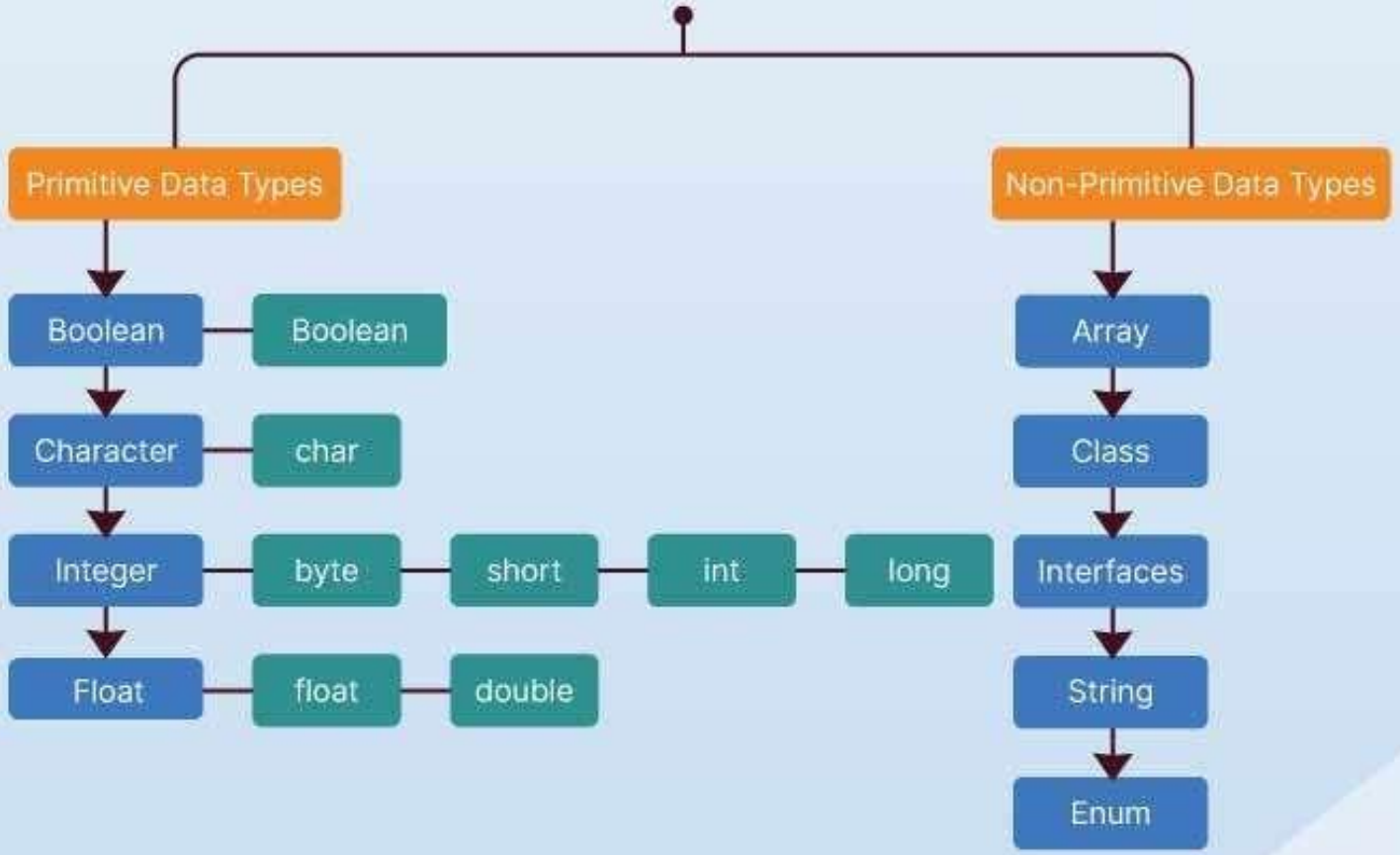
Bytecode


Javac.exe
Java.exe
Java Source Code .java
Java "Compiler" (x86)
Java "Compiler" (PowerPC)
Java "Compiler" (SPARC)
Java Bytecode-- platform independent
.class
Java Interpreter (X86)
Java Interpreter (PowerPC)
Java Interpreter (SPARC)

A First Simple Program Example java

```
import java.util.*;
class Example
{
   public static void main(String args[])
   {
   System.out.println(“First Java program");
   }
}
```

Data Types in Java



Datatypes

1) Primitive Data Types :

These store actual values directly and have fixed sizes.

- byte: 1 byte, range -128 to 127 (useful for saving memory)
- short: 2 bytes, range -32,768 to 32,767
- int: 4 bytes, range -2,147,483,648 to 2,147,483,647 (most common for integers)
- long: 8 bytes, for very large integers (use L at end like 10000000000L)
- float: 4 bytes, decimal values (use f like 3.14f)
- double: 8 bytes, more precision than float (default for decimals)
- char: 2 bytes, stores a single character (Unicode) like 'A', '₹'
- boolean: stores true/false (size depends on JVM implementation)

Datatypes

2) Non-Primitive (Reference) Data Types

- These store the address/reference of objects in memory.
- String: e.g., "Hello"
- Arrays: e.g., int[] a = {1,2,3};
- Classes / Objects: e.g., Student s = new Student();
- Interfaces: e.g., Runnable r;

Quick difference:

Primitive → direct value, fixed size, faster

Reference → points to an object, can have methods, generally more flexible

Literals

Java Literals are syntactic representations of boolean, character, numeric, or string data.

Literals provide a means of expressing specific values in your program.

Types of literals are:

1. Integer Literals
2. Floating-Point Literals
3. Boolean Literals
4. Character Literals

Integers

Datatype	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating-Point Types

Data type	Size	Description
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits

Operators

1) Arithmetic Operators:

Used for mathematical calculations.

+ (add), - (subtract), * (multiply), / (divide), % (modulus/remainder)

Example: c = a + b;

2) Unary Operators:

Work on a single operand.

+ (unary plus), - (unary minus)

++ (increment), -- (decrement)

! (logical NOT)

Example: a++; // increases a by 1

Operators

3) Relational (Comparison) Operators:

Used to compare two values, result is true/false.

==, !=, >, <, >=, <=

Example: a > b

4) Logical Operators:

Used with boolean conditions.

&& (AND), || (OR), ! (NOT)

Example: (a > 5 && b < 10)

Operators

5) Assignment Operators:

Used to assign/update values.

= , +=, -=, *=, /=, %=

Example: a += 5; // a = a + 5

6) Bitwise Operators:

Operate on bits (binary form).

&, |, ^ (XOR), ~ (NOT)

Example: a & b

7) Shift Operators:

Shift bits left/right.

<< (left shift), >> (right shift), >>> (unsigned right shift)

Example: a << 2

Operators

8) Ternary Operator:

Short form of if-else.

condition ? value1 : value2

Example: max = (a > b) ? a : b;

9) instanceof Operator:

Checks object type.

obj instanceof ClassName

Example: s instanceof String

Operators

Operators are used to perform operations on variables and values.

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Bitwise Operators

Operator	Description	Example	Same as	Result	Decimal
&	AND - Sets each bit to 1 if both bits are 1	5 & 1	0101 & 0001	0001	1
|	OR - Sets each bit to 1 if any of the two bits is 1	5 | 1	0101 | 0001	0101	5
~	NOT - Inverts all the bits	~ 5	~0101	1010	10
^	XOR - Sets each bit to 1 if only one of the two bits is 1	5 ^ 1	0101 ^ 0001	0100	4

Bitwise Operators

Operator	Description	Example	Same as	Result	Decimal
<<	Zero-fill left shift - Shift left by pushing zeroes in from the right and letting the leftmost bits fall off	9 << 1	1001 << 1	0010	2
>>	Signed right shift - Shift right by pushing copies of the leftmost bit in from the left and letting the rightmost bits fall off	9 >> 1	1001 >> 1	1100	12
>>>	Zero-fill right shift - Shift right by pushing zeroes in from the left and letting the rightmost bits fall off	9 >>> 1	1001 >>> 1	0100	4

The ? Operator

ternary (three-way) operator

expression1 ? expression2 : expression3

Example:

```
String result = (marks > 40) ? "pass" :
"fail";
```

Operator Precedence

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= +, = -, = *, = /, = %, = &, = ^, = |, = <<, = >>, = >>>

if Statement

The Java if statement tests the condition. It executes the if block if condition is true.

Syntax:

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

if Statement Example

```
class IfStatement
{
    public static void main(String[] args) {
        int number = 10;
        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        System.out.println("Statement outside if
block");
    }
}
```

If-else Statement

The Java if-else statement also test the condition. It executes the if block if condition is true otherwise else block is executed.

Syntax:

```
if(condition)
{
        //code to be executed
}
else
{
        //code to be executed when condition fails
}
```

If-else Statement Example

```
class Main {
    public static void main(String[] args) { int number = 10;
        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        // execute this block if number is not greater than else {
            System.out.println("The number is not positive.");
        }
        System.out.println("Statement outside if...else block");
    }
}
```

The if-else-if Ladder

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
        //code to be executed if condition1 is true
}else if(condition2){
        //code to be executed if condition2 is true
}
else if(condition3){
        //code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

if-else-if Ladder Example

```
class Main {
    public static void main(String[] args) {
        int number = 0;
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        else if (number < 0) {
            System.out.println("The number is negative.");
        }
        else {
            System.out.println("The number is 0.");
        }
    }
}
```

Switch case

The Java switch statement executes one statement from multiple **conditions, and its like if-else-if ladder statement.**

The switch statement works with byte, short, int, long, enum types, string.

Syntax:

```
switch(expression){
        case value1:
    //code to be executed; break;    //optional
  case value2:

        //code to be executed; break;
                //optional
  default:
   code to be executed if all cases are not mtched.
}
```

Switch case Example

```
class Main
{
    public static void main(String[] args)
      {
        int expression = 2;
        // switch statement to check size
        switch (expression) {
            case 1:
                System.out.println("Case 1");
                // matching case
            case 2:
                System.out.println("Case 2");
            case 3:
                System.out.println("Case 3"); default:
                System.out.println("Default case");
        }
    }
}
```

while

The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.

Syntax:

```
while(condition)
{
    //code to be executed
}
```

while Example

```
class Main {
    public static void main(String[] args) {
        // declare variables
        int i = 1, n = 5;
        // while loop from 1 to 5
        while(i <= n) {
            System.out.println(i);
            i++;
        }
    }
}
```

do-while

If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax:

```
do
{
        // body of loop
} while (condition);
```

do-while Example

```
class Main
  {
    public static void main(String[] args)
    {
        int i = 1, n = 5;
        do {
            System.out.println(i);
            i++;
        } while(i <= n);
    }
}
```

for loop

The Java for loop is a control flow statement that iterates a part of the programs multiple times.

If the number of iteration is fixed, it is recommended to use for loop.

Syntax:

```
for(initialization; condition; iteration)
{
        // body
}
```

for loop Example

```
class Main
{
    public static void main(String[] args)
    {
        int n = 5;
        // for loop
        for (int i = 1; i <= n; ++i) {
            System.out.println("Java is fun");
        }
    }
}
```

break

When a break statement is encounteredinside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java break statement is used to break loop or switch statement.

It can be used as a “civilized” form of goto.

break Example

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


        {
          for (int i = 1; i <= 10; ++i) {
          if (i == 5) {
                                break;
                        }
                        System.out.println(i);
                }
        }
}
```

continue

The Java **continue** statement is used to continue the loop.

It continues the current flow of the program and skips the remaining code at the specified condition.
In case of an inner loop, it continues the inner loop only.

while and do-while:
Control transferred directly to the conditional expression

In a **for** loop Control goes first to the iteration portion

continue Example

```
class Main
{
    public static void main(String[] args)
    {
        // for loop
        for (int i = 1; i <= 10; ++i) {
              // if value of i is between 4
              // and 9 continue is executed
            if (i == 4 || i == 9)
            {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

return

- A return statement causes the program control to transfer back to the caller of a method.
- Every method in Java is declared with a return type and it is mandatory for all java methods.
- When used, the program control transferred back to the caller of the method.