



# Introduction to Programming with Java





Talent Accelerator  
powered by 

# Agenda

- What is Programming?
- Why Java?
- Setting up Java & IDE
- Basic Java Syntax & Structure
- Data Types & Variables
- Operators & Expressions
- Control Flow Statements
- Functions & Methods



# What is Programming?

Programming is the process of creating a set of instructions for a computer to follow in order to perform a specific task or achieve a desired result.

It involves writing code in a programming language, testing that code, and debugging it to ensure it functions as expected.

Think of programming like cooking:

- The programming language is the ingredients.
- The code is the recipe.
- The computer is the chef that follows the recipe.



# History of Programming: Early days

- The concept of programming dates back to ancient civilizations.
- Algorithms used for calculations, such as the Euclidean algorithm, existed over 2300 years ago.
- Ada Lovelace in the 19th century is considered by many as the world's first programmer. She worked on Charles Babbage's Analytical Engine and wrote what is considered the first algorithm intended for implementation on a computer.



Talent Accelerator  
powered by 

# History of Programming: Modern Era

- 1940s-1950s: First high-level programming languages like Fortran, LISP, and COBOL.
- 1970s: C language developed, which later influenced Java, C++, and many others.
- 1980s-1990s: Rise of Object-Oriented Programming with languages like Java and Python.
- 2000s-Present: Evolution of web-based, mobile, and cloud-native applications. Languages like JavaScript, Swift, and Go gain prominence.



Talent Accelerator  
powered by 

# How Computers Interpret Instructions

- At its core, a computer understands only binary: 1s and 0s.
- The code we write in high-level languages (like Java, Python) is far from this binary form.
- For a computer to understand and execute our high-level code, it must be translated or interpreted into machine code.



# Compilation vs Interpretation

- *Compiled Languages*: Code is transformed into machine code before it is executed. The compiled output can then be run multiple times. Examples: C, C++, Java (to bytecode).
- *Interpreted Languages*: Code is translated into machine code on-the-fly, line by line, as it's executed. Examples: Python, JavaScript



**Why Java?**





Talent Accelerator  
powered by 

# Java Origins

Computer language innovation and development occurs for two fundamental reasons:

- 1) to adapt to changing environments and uses
- 2) to implement improvements in the art of programming

The development of Java was driven by both in equal measures.

Many Java features are inherited from the earlier languages:

$B \rightarrow C \rightarrow C++ \rightarrow \text{Java}$



Talent Accelerator  
powered by 

# Before Java : C

Designed by Dennis Ritchie in 1970s.

Before C, there was no language to reconcile: ease-of-use versus power, safety versus efficiency, rigidity versus extensibility.

BASIC, COBOL, FORTRAN, PASCAL optimized one set of traits, but not the other.

C- structured, efficient, high-level language that could replace assembly code when creating systems programs.

Designed, implemented and tested by programmers, not scientists.



# Before Java : C++

Designed by Bjarne Stroustrup in 1979.

Response to the increased complexity of programs and respective improvements in the programming paradigms and methods:

- 1) assembler languages
- 2) high-level languages
- 3) structured programming
- 4) object-oriented programming

(OOP) OOP – methodology that helps organize complex programs through the use of inheritance, encapsulation and polymorphism.

C++ extends C by adding object-oriented features.



Talent Accelerator  
powered by 

# Java History

Designed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems in 1991.

The original motivation is not Internet: platform-independent software embedded in consumer electronics devices.

With Internet, the urgent need appeared to break the fortified positions of Intel, Macintosh and Unix programmer communities.

Java as an “Internet version of C++”? No.

Java was not designed to replace C++, but to solve a different set of problems. There are significant practical/philosophical differences.



Talent Accelerator  
powered by 

# Java Technology

There is more to Java than the language.

Java Technology consists of:

- 1) Java Programming Language
- 2) Java Virtual Machine (JVM)
- 3) Java Application Programming Interfaces (APIs)



Talent Accelerator  
powered by 

# Java Language features

- 1) simple
- 2) object-oriented
- 3) robust
- 4) multithreaded
- 5) architecture-neutral
- 6) interpreted and high-performance
- 7) distributed
- 8) dynamic
- 9) secure



Talent Accelerator  
powered by 

# Java Language features(1)

- 1) simple – Java is designed to be easy for the professional programmer to learn and use.
- 2) object-oriented – a clean, usable, pragmatic approach to objects, not restricted by the need for compatibility with other languages.
- 3) robust – restricts the programmer to find the mistakes early, performs compile-time (strong typing) and run-time (exception-handling) checks, manages memory automatically.



## Java Language features(2)

4) multithreaded – supports multi-threaded programming for writing program that perform concurrent computations

5) architecture-neutral – Java Virtual Machine provides a platform independent environment for the execution of Java bytecode

6) interpreted and high-performance – Java programs are compiled into an intermediate representation – bytecode:

- a) can be later interpreted by any JVM

- b) can be also translated into the native machine code for efficiency.





## Java Language features(3)

7) distributed – Java handles TCP/IP protocols, accessing a resource through its URL much like accessing a local file.

8) dynamic – substantial amounts of run-time type information to verify and resolve access to objects at run-time.

9) secure – programs are confined to the Java execution environment and cannot access other parts of the computer.



# Java Program execution

Java programs are both compiled and interpreted:

Steps:

- write the Java program
- compile the program into bytecode
- execute (interpret) the bytecode on the computer through the Java

Virtual Machine

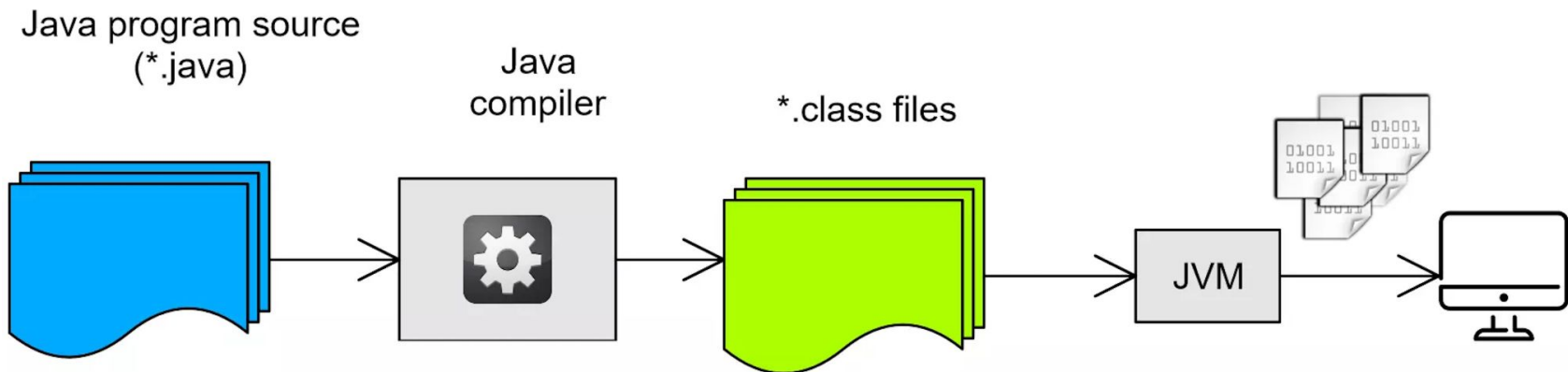
Compilation happens once.

Interpretation occurs each time the program is executed.



Talent Accelerator  
powered by 

# Java execution process





Talent Accelerator  
powered by 

# Java API

What is Java API?

- 1) a large collection of ready-made software components that provide many useful capabilities, e.g. graphical user interface
- 2) grouped into libraries (packages) of related classes and interfaces
- 3) together with JVM insulates Java programs from the hardware and operating system variations



# Java Program Types

Types of Java programs:

- 1) applications – standalone (desktop) Java programs, executed from the command line, only need the Java Virtual Machine to run
- 2) applets – Java program that runs within a Java-enabled browser, invoked through a “applet” reference on a web page, dynamically downloaded to the client computer
- 3) servlets – Java program running on the web server, capable of responding to HTTP requests made through the network
- 4) Enterprise Applications - Enterprise applications are large-scale applications built using Java EE technologies. These applications often involve distributed systems, databases, messaging, and more. Java EE provides components like EJBs (Enterprise JavaBeans) and web services for creating such applications.
- 5) API Libraries and Frameworks: Java programs can be developed as libraries or frameworks that provide functionality to other applications. These libraries can be integrated into various projects to enhance their capabilities.
- 6) etc.



Talent Accelerator  
powered by 

# Java Platform Features 1

1) **essentials** - objects, strings, threads, numbers, input/output, data structures, system properties, date and time, and others.

2) **networking**:

1) Hypertext Transfer Protocol (HTTP)

2) Transmission Control Protocol (TCP)

3) Internet Protocol (IP) addresses

3) **internationalization** - programs that can be localized for users worldwide, automatically adapting to specific locales and appropriate languages.



Talent Accelerator  
powered by 

## Java Platform Features 2

- 4) **security** – low-level and high-level security, including electronic signatures, public and private key management, access control, and certificates
- 5) **software components** – JavaBeans can plug into an existing component architecture
- 6) **object serialization** - lightweight persistence and communication, in particular using Remote Method Invocation (RMI)
- 7) **Java Database Connectivity (JDBC)** - provides uniform access to a wide range of relational databases



**Break 10 minutes**

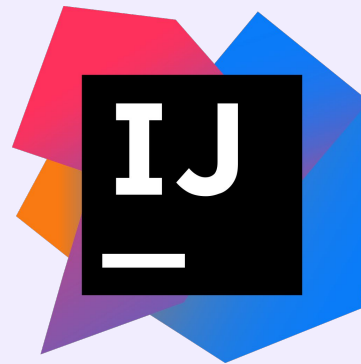




## Setting up Java & IDE



# Setting up Java & IDE



# Installing Java Development Kit (JDK)

JDK(Java Development Kit) contains the software and tools that is required to compile, debug and run java applications.

Latest version is JDK 20. Java 8, Java 11 and Java 17 are the three long-term support versions recommended by Oracle.

The JDK can be installed on the following platforms:

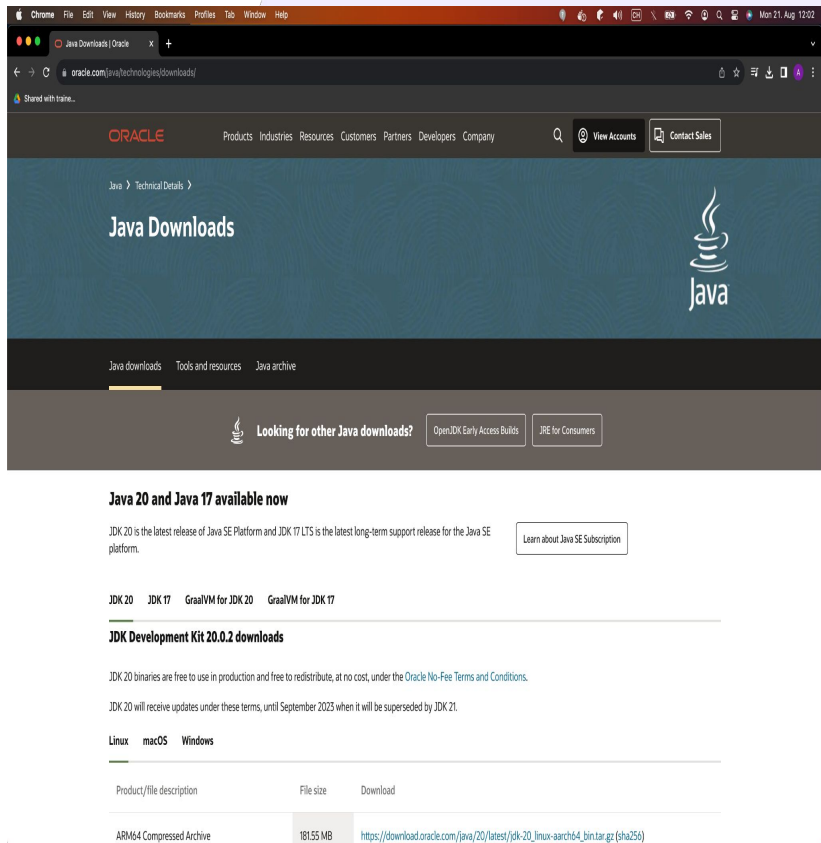
- [Microsoft Windows](#)
- [Linux](#)
- [macOS](#)

JDK can be downloaded from :

<https://www.oracle.com/java/technologies/downloads/>

In the following link you have instructions step by step to install JDK:

<https://docs.oracle.com/en/java/javase/20/install/overview-jdk-installation.html#GUID-8677A77F-231A-40F7-98B9-1FD0B48C346A>



The screenshot shows the Oracle Java Downloads page in a web browser. The page has a dark header with the Oracle logo and navigation links. The main content area is titled 'Java Downloads' and features a large Java logo. Below this, there are tabs for 'Java downloads', 'Tools and resources', and 'Java archive'. A section titled 'Java 20 and Java 17 available now' provides information about the latest releases. Below this, there is a table of downloads for JDK 20.0.2, including a table with columns for Product/file description, File size, and Download link.

Product/file description	File size	Download
ARM64 Compressed Archive	181.55 MB	<a href="https://download.oracle.com/java/20/latest/jdk-20_linux-aarch64_bin.tar.gz">https://download.oracle.com/java/20/latest/jdk-20_linux-aarch64_bin.tar.gz</a>



# Setting JAVA\_HOME & PATH Environment

Setting up **JAVA\_HOME** and **PATH** environment variables is necessary for the Java tools and applications to work properly from any directory.

Steps to carry out:

- 1) **JAVA\_HOME** variable should point to the top installation directory of Java environment.
- 2) **PATH** should point to the **bin** directory where the Java Virtual Machine - interpreter (java), compiler (javac) and other executables are located.



# Setting JAVA\_HOME & PATH Environment



Talent Accelerator  
powered by 

With more than one JVM(Java Virtual Machine) installed, the one you wish to use must appear as the first one in the PATH variable. Example:

**Windows:** Add '%JAVA\_HOME%\bin' to the beginning of the 'PATH' variable.

**Linux:** Include 'export PATH="\$PATH:\$JAVA\_HOME/bin"' in your '/etc/profile' or '.bashrc' files.

**macOS:** In your '~/.bash\_profile' or '~/.zshrc' file (depending on your shell), add:

```
bash
```

```
export JAVA_HOME=/path/to/jdk
export PATH=$JAVA_HOME/bin:$PATH
```

## To Test the Environment (Windows/Linux/macOS):

- Open a command prompt, terminal window, or Terminal app.
- Run `java -version`.
- The current version number of the installed JVM should appear.



# Basic Java Syntax



Talent Accelerator  
powered by 

# Java Syntax

On the most basic level, Java programs consist of :

- A. Whitespaces
- B. Identifiers
- C. Comments
- D. Literals
- E. Separators
- F. Keywords
- G. Operators

Each of them will be described in order.



# Whitespaces



Talent Accelerator  
powered by 

A whitespace is a space, tab or new line.

Java is a form-free language that does not require special indentation.

A program could be written like this :

```
class MyProgram {  
    Public static void main(String[] args){  
        System.out.println("Hello world!");  
    }  
}
```

It could be also written like this:

```
class MyProgram {Public static void main(String[] args){  
    System.out.println("Hello world!");}}
```



# Identifiers

Java identifiers:

- a) used for class names, method names, variable names
- b) an identifier is any sequence of letters, digits, “\_” or “\$” characters that do not begin with a digit
- c) Java is case sensitive, so `value`, `Value` and `VALUE` are all different.

Seven identifiers in this program:

```
class MyProgram {  
    public static void main(String[] args) {  
        System.out.println("First Java program.");  
    }  
}
```



# Comments



Talent Accelerator  
powered by 

Three kinds of comments:

1) Ignore the text between `/*` and `*/`:

```
/* text */
```

2) Documentation comment (`javadoc` tool uses this kind of comment to automatically generate software documentation):

```
/** documentation */
```

3) Ignore all text from `//` to the end of the line:

```
// text
```

```
/**
```

```
*
```

```
MyProgram implements application that displays
```

```
* a simple message on the standard output  
device.
```

```
*/
```

```
class MyProgram {
```

```
/* The main method of the class.*/
```

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

```
    //display string
```

```
    System.out.println("First Java program."); }}
```



# Literals

A literal is a constant value of certain type.

It can be used anywhere values of this type are allowed.

Examples:

- A. 100
- B. 56.6
- C. 'Y'
- D. "Test"

```
class MyProgram {  
    public static void main(String[] args) {  
        System.out.println("My first Java program.");  
    }  
}
```



# Separators

( )	parenthesis	lists of parameters in method definitions and invocations, precedence in expressions, type casts
{ }	braces	block of code, class definitions, method definitions, local scope, automatically initialized arrays
[ ]	brackets	declaring array types, referring to array values
;	semicolon	terminating statements, chain statements inside the “for” statement
,	comma	separating multiple identifiers in a variable declaration
.	period	separate package names from subpackages and classes, separating an object variable from its attribute or method



# Keywords

Keywords are reserved words recognized by Java that cannot be used as identifiers. Java defines 49 keywords as follows:

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



Talent Accelerator  
powered by 



**Break 10 minutes**



# Data Types & Variables



# Strong Typing

Java is a strongly-typed language:

- a) every variable and expression has a type
- b) every type is strictly defined
- c) all assignments are checked for type-compatibility
- d) no automatic conversion of non-compatible, conflicting types
- e) Java compiler type-checks all expressions and parameters
- f) any typing errors must be corrected for compilation to succeed





# Simple Types

Java defines eight simple types:

- 1) byte – 8-bit integer type
- 2) short – 16-bit integer type
- 3) int – 32-bit integer type
- 4) long – 64-bit integer type
- 5) float – 32-bit floating-point type
- 6) double – 64-bit floating-point type
- 7) char – symbols in a character set
- 8) boolean – logical values *true* and *false*



## Simple Type : byte

8-bit integer type.

Range: -128 to 127.

Example:

*byte b = -15;*

Usage: particularly when working with data streams.



Talent Accelerator  
powered by 

## Simple Type : short

16-bit integer type.

Range: -32768 to 32767.

Example:

*short c = 1000;*

Usage: probably the least used simple type.



## Simple Type : int

32-bit integer type.

Range: -2147483648 to 2147483647.

Example:

```
int b = -50000;
```

Usage:

- 1) Most common integer type.
- 2) Typically used to control loops and to index arrays.
- 3) Expressions involving the byte, short and int values are promoted to int before calculation.

## Simple Type : long

64-bit integer type.

Range: -9223372036854775808 to 9223372036854775807.

Example:

```
long l = 1000000000000000000L;
```

Usage:

- 1) useful when int type is not large enough to hold the desired value



Talent Accelerator  
powered by





## Simple Type : float

32-bit floating-point number.

Range:  $1.4e-045$  to  $3.4e+038$ .

Example:

*float f = 1.5;*

Usage:

- 1) fractional part is needed
- 2) large degree of precision is not required

## Simple Type : double

64-bit floating-point number.

Range:  $4.9e-324$  to  $1.8e+308$ .

Example:

*double pi = 3.1416;*

Usage:

- 1) accuracy over many iterative calculations
- 2) manipulation of large-valued numbers



Talent Accelerator  
powered by





## Simple Type : char

16-bit data type used to store characters.

Range: 0 to 65536.

Example:

```
char c = 'a';
```

Usage:

- 1) Represents both ASCII and Unicode character sets; Unicode defines a character set with characters found in (almost) all human languages.
- 2) Not the same as in C/C++ where char is 8-bit and represents ASCII only.

## Simple Type : boolean

Two-valued type of logical values.

Range: values *true* and *false*.

Example:

```
boolean b = (1<2);
```

Usage:

- 1) returned by relational operators, such as `1<2`
- 2) required by branching expressions such as *if* or *for*



Talent Accelerator  
powered by





# Variables

Java uses variables to store data.

To allocate memory space for a variable JVM(Java Virtual Machine) requires:

- 1) to specify the data type of the variable
- 2) to associate an identifier with the variable
- 3) optionally, the variable may be assigned an initial value

All done as part of variable declaration.



# Basic Variable Declaration

Basic form of variable declaration:

Variable data type must be one of the following:  
A simple data type  
A user defined data type called class type

Optional initial value

dataType

identifier

[ = value ] ;

Variable identifier must:  
Confirm to identifier rules



Talent Accelerator  
powered by 

# Variable Declaration

We can declare several variables at the same time:

```
type identifier [=value][, identifier [=value] ...];
```

Examples:

```
int a, b, c;
```

```
int d = 3, e, f = 5;
```

```
byte hog = 22;
```

```
double pi = 3.14159;
```

```
char kat = 'x';
```





Talent Accelerator  
powered by 

# Constant Declaration

A variable can be declared as final:

```
final double PI = 3.14;
```

The value of the final variable cannot change after it has been initialized:

```
PI = 3.13;
```



# Variable Identifiers

Identifiers are assigned to variables, methods and classes.

An identifier:

- 1) starts with a letter, underscore `_` or dollar `$`
- 2) can contain letters, digits, underscore or dollar characters
- 3) it can be of any length
- 4) it must not be a keyword (e.g. `class`)
- 5) it must be unique in its scope

Examples:     *identifier*,     *userName*,     *\_sys\_var1*,     *\$change*

The code of Java programs is written in Unicode, rather than ASCII, so letters and digits have considerably wider definitions than just a-z and 0-9.



# Naming Conventions

Conventions are not part of the language.

Naming conventions:

- 1) variable names begin with a lowercase letter
- 2) class names begin with an uppercase letter
- 3) constant names are all uppercase

If a variable name consists of more than one word, the words are joined together, and each word after the first begins with an uppercase letter.

The underscore character is used only to separate words in constants, as they are all caps and thus cannot be case-delimited.



# Variable Initialization

During declaration, variables may be optionally initialized.

Initialization can be static or dynamic:

1) static initialize with a literal:

```
int n = 1;
```

2) dynamic initialize with an expression composed of any literals, variables or method calls available at the time of initialization:

```
int m = n + 1;
```

The types of the expression and variable must be the same.



# Operators & Expressions



Talent Accelerator  
powered by 

# Operators Type

Java operators are used to build value expressions.

Java provides a rich set of operators:

- 1) assignment
- 2) arithmetic
- 3) relational
- 4) logical
- 5) bitwise
- 6) etc



# Operators and Operands

Each operator takes one, two or three operands:

- 1) a unary operator takes one operand

```
j++;
```

- 2) a binary operator takes two operands

```
i = j++;
```

- 3) a ternary operator requires three operands

```
i = (i>12) ? 1 : i++;
```



# Assignment Operators

A binary operator:

```
variable = expression;
```

It assigns the value of the expression to the variable.

The types of the variable and expression must be compatible.

The value of the whole assignment expression is the value of the expression on the right, so it is possible to chain assignment expressions as follows:

```
int x, y, z;
```

```
x = y = z = 2;
```





# Arithmetic Operators

Java supports various arithmetic operators for:

- 1) integer numbers
- 2) floating-point numbers

There are two kinds of arithmetic operators:

- 1) basic: addition, subtraction, multiplication, division and modulo
- 2) shortcut: arithmetic assignment, increment and decrement



## Table: Basic Arithmetic Operators

+	$op1 + op2$	Adds $op1$ and $op2$
-	$op1 - op2$	Subtract $op1$ and $op2$
*	$op1 * op2$	Multiplies $op1$ and $op2$
/	$op1 / op2$	Divides $op1$ and $op2$
%	$op1 \% op2$	Computes the remainder of dividing $op1$ by $op2$



# Arithmetic Assignment Operators

Instead of writing

`variable = variable operator expression;`

for any arithmetic binary operator, it is possible to write shortly

`variable operator= expression;`

Benefits of the assignment operators:

- 1) save some typing
- 2) are implemented more efficiently by the Java run-time system



## Table: Arithmetic Assignments

+=	V += expr;	V = v + expr;
-=	v-=expr;	V = v - expr;
*=	v*=expr;	V = v * expr;
/=	v/=expr;	V = v / expr;
%=	v%=expr;	V = v % expr;



# Increment/Decrement Operators

Two unary operators:

- 1) ++ increments its operand by 1
- 2) -- decrements its operand by 1

The operand must be a numerical variable.

Each operation can appear in two versions:

- **prefix** version evaluates the value of the operand after performing the increment/decrement operation
- **postfix** version evaluates the value of the operand before performing the increment/decrement operation



## Table: Increment/Decrement

++	v++;	Return value of v, then increment v
++	++v;	Increment v, then return its value
--	v--;	Return value of v, then decrement v
--	--v;	Decrement v , then return its value



Talent Accelerator  
powered by 

# Relation Operators

Relational operators determine the relationship that one operand has to the other operand, specifically equality and ordering.

The outcome is always a value of type boolean.

They are most often used in branching and loop control statements.



## Table: Relational Operators

==	Equals to	Apply to any type
!=	Not equal to	Apply to any type
>	Greater than	Apply to numerical types only
<	Less than	Apply to numerical types only
>=	Greater than or equal	Apply to numerical types only
<=	Less than or equal	Apply to numerical types only





# Logical Operators

Logical operators act upon boolean operands only.

The outcome is always a value of type boolean.

In particular, “and” and “or” logical operators occur in two forms:

- 1) full `op1 & op2` and `op1 | op2` where both `op1` and `op2` are evaluated
- 2) short-circuit - `op1 && op2` and `op1 || op2` where `op2` is only evaluated if the value of `op1` is insufficient to determine the final outcome



## Table: Logical Operators

&	Op1 & op2	AND
	Op1   op2	OR
&&	Op1 && op2	Short-circuit AND
	Op1    op2	Short-circuit OR
!	! op	NOT
^	Op1 ^ op2	XOR



# Bitwise Operators

Bitwise operators apply to integer types only.

They act on individual bits of their operands.

There are three kinds of bitwise operators:

- 1) basic bitwise AND, OR, NOT and XOR
- 2) shifts left, right and right-zero-fill
- 3) assignments bitwise assignment for all basic and shift operators



# Bitwise Operators

~	op~	Inverts all bits of its operand
&	Op1 & op2	Produces 1 bit if both operands are 1
	Op1   op2	Produces 1 bit if either operands are 1
^	Op1 ^ op2	Produces 1 bit if exactly one operands is 1
>>	Op1 >> op2	Shift all bits in op1 right by the value of op2
<<	Op1 << op2	Shift all bits in op1 left by the value of op2
>>>	Op1 >>> op2	Shift op1 right by op2 value, write zero on the left



Talent Accelerator  
powered by 

# Type of Expressions in Java

An expression in Java is a series of operators, variables, and method calls constructed according to the syntax of the language for evaluating a single value.

A simple expression in Java has a type that can either be a primitive type or a reference type.

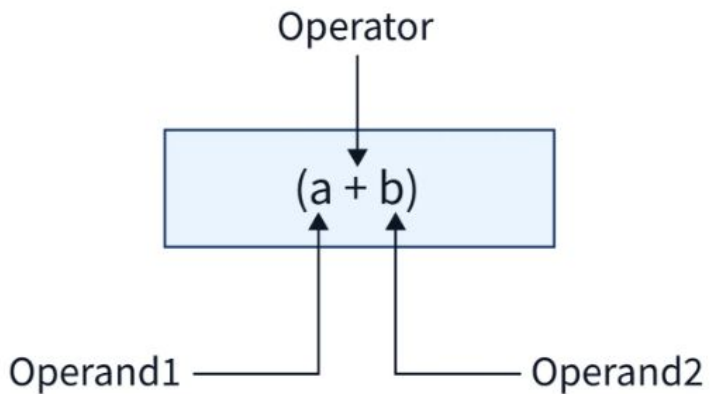
There are 3 types of expressions in Java:

- Infix Expression
- Postfix Expression
- Prefix Expression



# Infix Expression

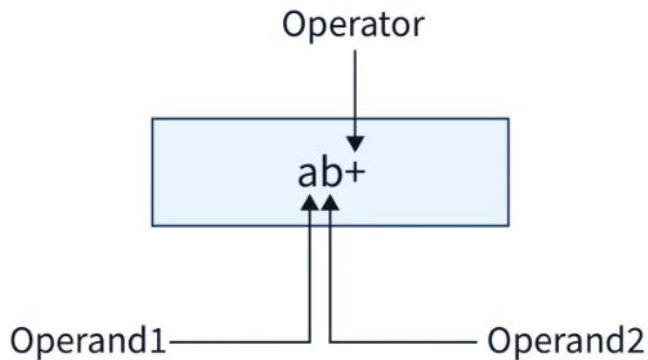
It is an expression in which the operators are used between operands.





# Postfix Expression

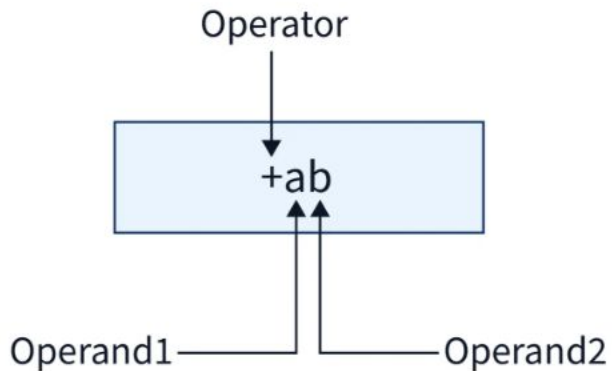
It is an expression in which operators are used after operands.





# Prefix Expression

It is an expression in which operators are used before an operand.







# Control Flow Statements



Talent Accelerator  
powered by 

# Control Flow

Writing a program means typing statements into a file.

Without control flow, the interpreter would execute these statements in the order they appear in the file, left-to-right, top-down.

Control flow statements, when inserted into the text of the program, determine in which order the program should be executed.



# Control Flow Statements

Java control statements cause the flow of execution to advance and branch based on the changes to the state of the program.

Control statements are divided into three groups:

- 1) selection statements allow the program to choose different parts of the execution based on the outcome of an expression
- 2) iteration statements enable program execution to repeat one or more statements
- 3) jump statements enable your program to execute in a non-linear fashion



# Selection Statement

Java selection statements allow to control the flow of program's execution based upon conditions known only during run-time.

Java provides four selection statements:

- 1) if
- 2) if-else
- 3) if-else-if
- 4) switch



**Break 10 minutes**



## Functions & Methods



# Methods

A method in Java is a subroutine that is part of a *class*. The subroutine is like a miniature program that can execute in other parts of the program. Methods promote code reuse and maintainability.

A *method definition* consists of the modifier, return type, name, parameter list, exception list, and body. The method name and parameter types form the *method signature*. The method signature uniquely identifies the method for execution.

Note: Java also has *constructor methods*. A constructor is a special method that creates an object of a class.

There are two types of methods: *procedures* and *functions*.



# Procedures and Functions

A procedure is a method that *does not* have a return value. To define a method to be a procedure, define the return type to be void. An example of a built-in procedure in Java is `System.out.println()`. This procedure simply outputs its parameter to the console, without returning a value

A function is a method that *does* have a return value. To define a method to be a function, set its return type to be the type of the value it is returning. An example of a built-in function in Java is `Math.pow()`. This Math function accepts two double parameters and returns the first parameter raised to the power of the second parameter. The return type is double.





Questions?





**Thank you for your attention!**