

UNIT 2

Tokens, Expressions and Control Structures

(5 Hours) — Object Oriented Programming in Java (CACS2O4)

Every Java program is built from a small set of basic building blocks: the smallest individual pieces of source code, called tokens; the values and variables that these tokens represent; and the expressions and control structures that combine them into working logic. This unit covers Java's data types, variables and constants, type conversion, arrays, operators, and the branching, looping, and jumping statements used to control the flow of a program.

2.1 Tokens

A token is the smallest individual unit of a Java program that the compiler can recognize. The compiler breaks every source file into a stream of tokens before it does anything else. Java defines five kinds of tokens:

Token Type	Description	Example
Identifiers	Names given to classes, methods, and variables	total, MyClass, count1
Keywords	Reserved words with special meaning to the compiler	class, if, for, public
Literals	Fixed values written directly in the code	10, 3.14, 'A', "Hi", true
Operators	Symbols that perform operations on operands	+, -, ==, &&, =
Separators/Punctuators	Symbols that organize and separate code	{ }, (), ;, ,

Identifiers

An identifier is the name used for a class, method, variable, or other user-defined item. Java identifiers must begin with a letter, an underscore (`_`), or a dollar sign (`$`) and may be followed by any combination of letters, digits, underscores, or dollar signs. Identifiers are case-sensitive (total and Total are different), cannot be a reserved keyword, and cannot contain spaces.

Literals

A literal is a value that is written directly into the source code, as opposed to a value stored in a variable. Java supports integer literals (10, 0x1A, 010, 0b1010), floating-point literals (3.14, 2.5f), character literals ('A', '\n'), string literals ("Hello"), and boolean literals (true, false).

Comment Syntax

Comments are ignored by the compiler and exist purely to document code for human readers. Java supports three comment styles:

```
// Single-line comment

/* Multi-line
   comment */

/** Documentation comment (used by the javadoc tool)
 *  @param n the number to square
 */
```

2.2 Primitive Data Types

Java is a strongly typed language — every variable must be declared with a data type before it is used. Java defines eight primitive types, which store simple values directly rather than references to objects: four integer types, two floating-point types, a character type, and a boolean type.

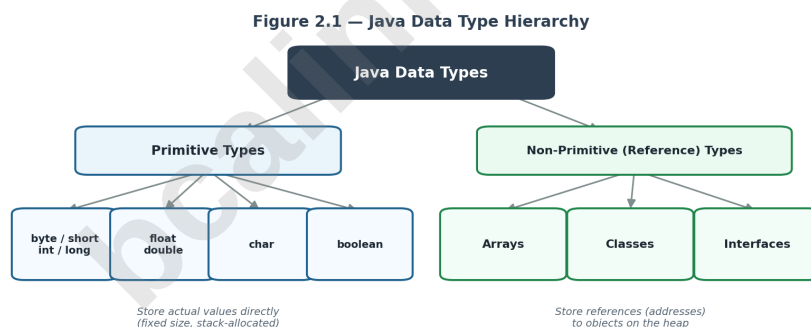


Figure 2.1 — Primitive types store values directly; non-primitive (reference) types store an address pointing to an object on the heap.

Integer Types

Type	Size	Range	Example
byte	1 byte (8 bit)	−128 to 127	byte b = 100;
short	2 bytes (16 bit)	−32,768 to 32,767	short s = 2000;
int	4 bytes (32 bit)	− 2.1 billion to 2.1 billion (approx.)	int n = 45000;
long	8 bytes (64 bit)	− 9.2×10^{18} to 9.2×10^{18} (approx.)	long L = 9999999999L;

Floating-Point Types

Type	Size	Precision	Example
float	4 bytes (32 bit)	~6–7 significant decimal digits	float f = 3.14f;
double	8 bytes (64 bit)	~15 significant decimal digits	double d = 3.14159265;

Characters and Booleans

The char type holds a single 16-bit Unicode character (range 0 to 65,535) and is written in single quotes, e.g. char grade = 'A';. The boolean type holds only one of two logical values, true or false, and is commonly used to control conditions and loops — Java, unlike C, does not allow integers to be used as booleans.

Note: All primitive numeric types in Java are signed (they can represent negative values) — Java has no unsigned integer types, unlike languages such as C.

2.3 User-Defined Data Types

Beyond the eight primitives, Java lets programmers define their own reference types, built up from primitives and other objects. The most common user-defined types are classes (blueprints that bundle fields and methods together), interfaces (contracts specifying methods a class must implement), and arrays (fixed-size, indexed collections of elements of the same type — themselves a reference type even when they hold primitives). Enumerations (enum) are a further special-purpose user-defined type used to represent a fixed set of named constants, e.g. enum Day { MON, TUE, WED };

2.4 Declarations and Constants

A declaration introduces a variable's name and type to the compiler, for example int age; declares a variable named age of type int. A constant is a value that, once assigned, cannot be changed for the rest of the program. In Java, constants are created using the final keyword, conventionally written in uppercase:

```
final double PI = 3.14159;
final int MAX_STUDENTS = 60;
// PI = 3.0;    // compile-time error – cannot reassign a final variable
```

2.5 Type Conversion and Casting

Type conversion is the process of changing a value from one data type to another. Java supports two kinds of conversion:

Conversion	Direction	Performed by	Example
Widening (implicit)	Smaller type → larger type (e.g. int → double)	Compiler, automatically	double d = 100; // int widened to double
Narrowing (explicit casting)	Larger type → smaller type (e.g. double → int)	Programmer, using a cast	int n = (int) 9.7; // n becomes 9

Widening conversions are always safe, since the destination type can represent every value of the source type without loss. Narrowing conversions may lose information (a fractional part is truncated, not rounded, and very large values may overflow) and must therefore be requested explicitly using a cast, written as (targetType) value.

Example 2.1 — Casting

```
double price = 49.99;
```

```
int wholePart = (int) price; // wholePart = 49 (fraction discarded)
```

2.6 Variables: Definition, Assignment, and Default Initialization

A variable is a named storage location that holds a value of a particular type. Defining a variable reserves memory for it and associates it with a type and name; assignment stores a value into that memory using the = operator. The two steps can be combined:

```
int marks;      // declaration (definition)
marks = 78;     // assignment
int age = 20;   // declaration + assignment (initialization) in one line
```

Java distinguishes between local variables, declared inside a method (which must be explicitly initialized before use — Java gives them no default value), and instance/class fields, declared inside a class but outside any method, which are automatically given a default value if the programmer does not supply one:

Type	Default Value
byte, short, int, long	0
float, double	0.0

Type	Default Value
char	'\u0000' (null character)
boolean	false
Any reference type (object, array, String)	null

Note: Only fields have default values automatically. A local variable used before it is explicitly assigned a value causes a compile-time error in Java.

2.7 Command-Line Arguments

Java programs can receive input directly from the command line at the moment they are launched. These values are passed into the main() method's parameter, String[] args, where each word typed after the class name on the command line becomes one element of the args array.

```
public class Greet {
    public static void main(String[] args) {
        System.out.println("Hello, " + args[0] + "!");
    }
}
// Terminal:
// java Greet Alice
// Output: Hello, Alice!
```

2.8 Arrays of Primitive Data Types

An array is a fixed-size, indexed collection of elements that all share the same type. Once created, an array's size cannot change. Array indices in Java start at 0, and the number of elements is available via the length field.

```
int[] marks = new int[5];           // array of 5 ints, all default to 0
marks[0] = 78;
marks[1] = 85;

int[] scores = {90, 75, 88, 60, 95}; // declare and initialize together
System.out.println(scores.length);   // 5
System.out.println(scores[2]);       // 88
for (int i = 0; i < scores.length; i++) {
    System.out.println(scores[i]);
}
```

Note: Accessing an index outside the valid range (0 to length-1) throws an `ArrayIndexOutOfBoundsException` at run time.

2.9 Garbage Collection

In languages such as C, programmers must manually free memory they no longer need. Java instead manages memory automatically through garbage collection: a background process within the JVM that periodically identifies objects on the heap that are no longer reachable from any active reference in the program, and reclaims the memory they occupy. An object becomes eligible for garbage collection once no variable anywhere in the program still refers to it — for example, after its reference is reassigned or set to null, or when it goes out of scope.

Key Points on Garbage Collection

- ✓ Garbage collection is automatic — Java programmers do not explicitly free memory
- ✓ An object is collected once it becomes unreachable from any live reference
- ✓ `System.gc()` only suggests that collection run; it does not force it immediately
- ✓ Automatic memory management removes an entire class of bugs, such as memory leaks caused by forgetting to free memory, and dangling pointers

2.10 Expressions and Operators

An expression is a combination of variables, literals, method calls, and operators that evaluates to a single value, for example $(a + b) * 2$. Java provides a rich set of operators for building expressions.

Arithmetic Operators

Operator	Meaning	Example (a=10, b=3)
+	Addition	$a + b \rightarrow 13$
-	Subtraction	$a - b \rightarrow 7$
*	Multiplication	$a * b \rightarrow 30$
/	Division	$a / b \rightarrow 3$ (integer division)
%	Modulus (remainder)	$a \% b \rightarrow 1$

Relational Operators

Operator	Meaning	Example (a=10, b=3)
==	Equal to	a == b → false
!=	Not equal to	a != b → true
>	Greater than	a > b → true
<	Less than	a < b → false
>=	Greater than or equal to	a >= b → true
<=	Less than or equal to	a <= b → false

Logical Operators

Operator	Meaning	Example (a=true, b=false)
&&	Logical AND (short-circuit)	a && b → false
	Logical OR (short-circuit)	a b → true
!	Logical NOT	!a → false

Bitwise and Shift Operators

Operator	Meaning	Example (a=6, b=3)
&	Bitwise AND	a & b → 2
	Bitwise OR	a b → 7
^	Bitwise XOR	a ^ b → 5
~	Bitwise complement (NOT)	~a → -7
<<	Left shift	a << 1 → 12
>>	Signed right shift	a >> 1 → 3
>>>	Unsigned right shift	-8 >>> 1 → 2147483644

Assignment Operators

Operator	Equivalent To	Example (a=10)
=	Simple assignment	a = 5
+=	a = a + value	a += 3 → 13
-=	a = a - value	a -= 3 → 7

Operator	Equivalent To	Example (a=10)
<code>*=</code>	<code>a = a * value</code>	<code>a *= 2 → 20</code>
<code>/=</code>	<code>a = a / value</code>	<code>a /= 2 → 5</code>
<code>%=</code>	<code>a = a % value</code>	<code>a %= 3 → 1</code>

Conditional (Ternary) Operator

The ternary operator, `?:`, is Java's only operator that takes three operands. It evaluates a boolean condition and returns one of two values, providing a compact alternative to a simple if-else:

```
int a = 10, b = 3;
int max = (a > b) ? a : b;    // max = 10
```

Auto-Increment and Auto-Decrement

The `++` and `--` operators increase or decrease a variable's value by 1. Each can be used in prefix form (`++x`), which changes the value before it is used in the surrounding expression, or postfix form (`x++`), which uses the original value first and changes it afterward.

```
int x = 5;
System.out.println(x++);    // prints 5, then x becomes 6
System.out.println(++x);    // x becomes 7, then prints 7
```

Note: Operator precedence determines which operator is evaluated first in a mixed expression (e.g. `*` and `/` bind tighter than `+` and `-`). Parentheses can always be used to make the intended order explicit and improve readability.

2.11 Control Statements

By default, statements in a Java program execute sequentially, one after another. Control statements let a program branch, repeat, or jump between different parts of its code based on conditions.

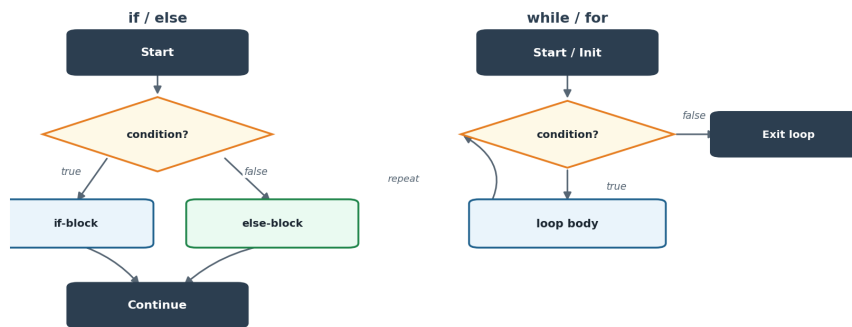


Figure 2.2 — Branching (if/else) evaluates one path; looping (while/for) repeats a body while a condition holds.

Branching Statements

The if statement executes a block only when its condition is true, and may be extended with else if and else to handle multiple alternatives:

```

if (marks >= 90) {
    grade = 'A';
} else if (marks >= 75) {
    grade = 'B';
} else {
    grade = 'C';
}

```

The switch statement selects one of several blocks to execute based on the value of a single expression, and is often clearer than a long if-else-if chain when comparing one variable against several constant values:

```

switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    default:
        System.out.println("Another day");
}

```

Note: Without a break statement, execution “falls through” into the next case — this is sometimes used deliberately, but is a common source of bugs when accidental.

Looping Statements

The while loop checks its condition before each iteration, so its body may run zero or more times; the do-while loop checks its condition after each iteration, so its body always runs at least once; and the for loop bundles initialization, condition, and update into a single compact header, making it well suited to loops with a known number of repetitions.

Loop	Condition Checked	Minimum Executions	Typical Use
while	Before each iteration	0	Repeat while a condition holds, count unknown in advance
do-while	After each iteration	1	Body must run at least once (e.g. menu prompts)
for	Before each iteration	0	Fixed or countable number of iterations

```
// while loop
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}

// do-while loop
int j = 0;
do {
    System.out.println(j);
    j++;
} while (j < 5);

// for loop
for (int k = 0; k < 5; k++) {
    System.out.println(k);
}
```

Jumping Statements

Statement	Effect
break	Immediately exits the nearest enclosing loop or switch statement
continue	Skips the rest of the current loop iteration and moves to the next one
return	Exits the current method immediately, optionally sending back a value to the caller

Example 2.2 — break and continue

```
for (int i = 1; i <= 10; i++) {  
    if (i == 6) break; // stop the loop entirely once i is 6  
    if (i % 2 == 0) continue; // skip printing even numbers  
    System.out.println(i);  
}  
// Output: 1 3 5
```

Key Takeaways

- ✓ Tokens are the smallest units of source code: identifiers, keywords, literals, operators, and separators
- ✓ Java has eight primitive types (byte, short, int, long, float, double, char, boolean) plus user-defined reference types (classes, interfaces, arrays, enums)
- ✓ Widening conversions happen automatically; narrowing conversions require an explicit cast and may lose data
- ✓ Fields get automatic default values (0, 0.0, false, null); local variables do not and must be initialized before use
- ✓ Arrays are fixed-size, zero-indexed collections; their length is fixed once created
- ✓ Garbage collection automatically reclaims memory occupied by objects that are no longer reachable
- ✓ Java's operators cover arithmetic, relational, logical, bitwise, shift, assignment, ternary, and increment/decrement needs
- ✓ if/switch branch program flow; while/do-while/for repeat it; break, continue, and return jump out of the normal flow

— End of Unit 2 —