Java Cheat Sheet

current release line — Java 25 LTS

COMPILE & RUN

```
javac Hello.java      # → Hello.class
java Hello            # run class
java Hello.java       # single-file source launch
jshell              # REPL: /vars /methods /exit
jar --create --file app.jar --main-class Hello -C out .
java -jar app.jar
java --enable-preview --release 25 ...
```

JDK = compiler + tools + JRE. Source → **bytecode** (`.class`) → JVM (class loader → verifier → interpreter + JIT). New feature release every 6 months; LTS every 2 yrs.

PRIMITIVES & VAR

byte short int long	8/16/32/64-bit signed integers
float double	32/64-bit IEEE-754
char	16-bit UTF-16 code unit
boolean	true / false

```
int m = 1_000_000; long b = 0b1010L; int h = 0xFF;
final double PI = 3.14159;          // constant
var list = new ArrayList<String>();  // inferred; locals only
int n = (int) 3.9;                  // narrowing cast → 3
Math.addExact(a, b);                // throws on overflow
```

Defaults: 0 / 0.0 / '' / false / null (fields only).

OPERATORS

```
+ - * /  %          / on ints truncates
== != < <= > >=      && || !  (short-circuit)
& | ^ ~  << >> >>>   cond ? a : b
= += -= *= ...      instanceof
```

Precedence (high→low): postfix `++ --` · unary · `* / %` · `+ -` · shifts · relational · `== !=` · `&` · `^` · `|` · `&&` · `||` · `?:` · assignment.

STRINGS & TEXT BLOCKS

```
String s = "hi"; s.length(); s.strip();
s.split(",");
s.isBlank(); s.repeat(3); s.chars();
"%d".formatted(4);
a.equals(b)             // value; == compares identity
var sb = new StringBuilder();
sb.append(x).append(y);
String j = String.join(", ", list);
String t = """
    line 1
    line 2""";          // text block: common indent stripped
```

CONTROL FLOW

```
if (c) {...} else {...}      while (c) {...}
do {...} while (c);
for (int i=0;i<n;i++) {...}   for (var x : xs) {...}
break; continue; outer: for(...) { break outer; }

// switch expression (arrow, exhaustive, yields a value)
String r = switch (day) {
    case MON, TUE -> "early";
    case WED -> "mid";
    default -> { yield "late"; }
};
```

ARRAYS

```
int[] a = new int[5];           int[] b = {1,2,3};
int[][] grid = new int[3][4];   int[][] jag = new int[3][];
a.length;  a[i];                // AIOOBE if out of range
Arrays.sort(a); Arrays.binarySearch(a, x);
Arrays.copyOf(a, len); Arrays.asList(b);
Arrays.stream(a);
void m(int... xs) { }           // varargs == int[]
```

CLASSES & OBJECTS

```
class Account {
  private long balance;              // encapsulate
  static final String BANK = "X";    // constant
  Account(long b) { this.balance = b; }
  Account() { this(0); }             // constructor chaining
  long balance() { return balance; }
}
var a = new Account(100);            // heap; GC frees it
static { /* class-init block */ }
```

Access: **private** < package-private (default) < **protected** < **public**.

INHERITANCE & OBJECT

```
class Save extends Account {
  Save(long b) { super(b); }
  @Override long balance() { return super.balance(); }
}
abstract class Shape { abstract double area(); }
final class Circle extends Shape { ... }   // no subclass
```

Object: `toString()` `equals(Object)` `hashCode()` `getClass()`. Override `equals`+`hashCode` together (`Objects.equals` / `Objects.hash`). Java is **pass-by-value** (the reference is copied).

INTERFACES

```
interface Repo<T> {
  void save(T t);
  default boolean isEmpty() { return count() == 0; }
  static Repo<?> empty() { ... }
  private void log() { ... }
}
class C implements Repo<X>, Serializable { ... }
// diamond: C.super.m() to disambiguate; class wins over iface
```

`Comparable<T>.compareTo` = natural order; `Comparator.comparing(f).thenComparing(g).reverse`

RECORDS & SEALED

```
record Point(int x, int y) {
  Point {                          // compact ctor: validate
    if (x < 0) throw new IllegalArgumentException();
  }
}
// auto: ctor, x(), y(), equals, hashCode, toString; final

sealed interface Expr permits Lit, Add {}
record Lit(int v) implements Expr {}
record Add(Expr l, Expr r) implements Expr {}
// non-sealed to reopen a branch
```

ENUMS

```
enum Dir {
  N(0), E(90), S(180), W(270);
  final int deg; Dir(int d){ deg = d; }
}
Dir.values(); Dir.valueOf("N"); d.name(); d.ordinal();
EnumSet.of(Dir.N, Dir.S);  EnumMap<Dir,String> m;
```

LAMBDAS & REFS

```
Runnable r = () -> System.out.println("go");
BiFunction<Integer,Integer,Integer> add = (a,b) -> a+b;
// method references (4 kinds):
Integer::parseInt      // static
str::length            // bound instance
String::toUpperCase    // unbound instance
ArrayList::new         // constructor
```

Captures **effectively final** vars; `this` = enclosing instance (unlike an anonymous class).

GENERICS

```
class Box<T> { T v; }
<T extends Comparable<T>> T max(List<T> xs) { ... }
List<? extends Number> src;   // producer → read
List<? super Integer> dst;    // consumer → write
// PECS: Producer Extends, Consumer Super
```

Erasure: no `new T[]`, no `T.class`, no `x instanceof T`; `@SafeVarargs` for generic varargs.

PATTERN MATCHING

```
if (o instanceof String s && !s.isBlank())
use(s);

String d = switch (shape) {
  case null             -> "none";
  case Circle c         -> "r=" + c.r();
  case Rect(var w, var h) -> w + "x" + h;   // deconstruction
  case Shape s when s.area() == 0 -> "empty"; // guard
  default               -> "?";
};   // exhaustive over a sealed type → no default needed
```

ANNOTATIONS

```
@Override @Deprecated(since="2", forRemoval=true)
@SuppressWarnings("unchecked")
@FunctionalInterface @SafeVarargs

@Retention(RUNTIME) @Target(METHOD)
@interface Timed { String value() default ""; }

m.getAnnotation(Timed.class);   // reflection: needs opens/RUNTIME
```

EXCEPTIONS

```
try (var in = Files.newBufferedReader(p)) {   // auto-close
  ...
} catch (IOException | RuntimeException e) {  // multi-catch
  throw new AppException("failed", e);        // chaining
} finally { ... }
```

Checked (extends Exception) must be caught/declared; **unchecked** (RuntimeException, Error) need not. `e.getCause()`, `e.getSuppressed()`, `assert cond;` (–ea).

OPTIONAL

```
Optional<User> u = repo.find(id);
u.map(User::name).filter(n -> !n.isBlank())
 .orElseGet(() -> "anon");
u.ifPresentOrElse(this::use, this::warn);
u.orElseThrow(() -> new NotFound(id));
// avoid: fields, params, bare get(), Optional<List>
```

COLLECTIONS

```
List<T>  ArrayList / LinkedList
Set<T>   HashSet / LinkedHashSet / TreeSet
Map<K,V> HashMap / LinkedHashMap / TreeMap
Deque<T> ArrayDeque   Queue PriorityQueue
List.of(1,2)  Map.of("a",1)  Map.entry(k,v)   // immutable
list.removeIf(x -> x==null);
map.getOrDefault(k, d);
map.computeIfAbsent(k, key -> new ArrayList<>());
```

Sequenced (21+): `getFirst/getLast/addFirst/addLast/reversed`. Keys need consistent `equals`/`hashCode`.

STREAMS & COLLECTORS

```
list.stream()
  .filter(x -> x > 0)
  .map(Object::toString)
  .sorted().distinct().limit(10)
  .collect(Collectors.groupingBy(String::length,

Collectors.counting()));
IntStream.range(0, n).sum();
list.stream().toList();
reduce(0, Integer::sum);  anyMatch / findFirst -> Optional
.parallel()  // large, CPU-bound, stateless only
```

Lazy until a terminal op; a stream is consumed once.

JAVA.UTIL.FUNCTION

Function<T,R>	apply; andThen / compose
Predicate<T>	test; and / or / negate
Consumer<T>	accept; andThen
Supplier<T>	get
UnaryOperator<T>	Function<T,T>
BiFunction<T,U,R>	+ Int/Long/Double specializations

NUMBERS & MATH

```
Integer.parseInt("42");  Integer.valueOf(1) == ... // cache −128..127
BigDecimal p = new BigDecimal("0.10");   // string ctor for money!
p.add(q); p.setScale(2, RoundingMode.HALF_UP);
BigInteger.valueOf(2).pow(64);
Math.floorMod(−1, 3);  Math.multiplyExact(a, b);
0.1 + 0.2 != 0.3;  Double.compare(a, b);  // NaN-safe
```

JAVA.TIME

```
LocalDate.now(); LocalDate.of(2025,1,31);
LocalDateTime dt; Instant.now();          // machine time
ZonedDateTime z = dt.atZone(ZoneId.of("Europe/Madrid"));
Duration.ofMinutes(90);  Period.ofDays(10);
DateTimeFormatter.ISO_DATE;  fmt.format(d);
LocalDate.parse(s, fmt);
d.plusDays(1);
d.with(TemporalAdjusters.lastDayOfMonth());
```

All immutable & thread-safe. Legacy: `Date.toInstant()`.

I/O & NIO.2

```
Path p = Path.of("dir", "file.txt");
String s = Files.readString(p);
Files.writeString(p, s);
List<String> ls = Files.readAllLines(p);
try (var lines = Files.lines(p)) { lines.forEach(...); }
Files.exists(p); Files.createDirectories(d);
Files.copy(a,b);
try (var w = Files.newBufferedWriter(p)) { w.write(s); }
new Scanner(System.in).nextLine();
```

REGEX

```
Pattern pat = Pattern.compile("(?<y>\\d{4})-(\\d{2})");
Matcher m = pat.matcher(text);
if (m.find()) { m.group("y"); m.group(1); }
"a,b,,c".split(",", −1);   s.matches("\\d+");
s.replaceAll("\\s+", " ");
m.replaceAll(r -> r.group().toUpperCase());
// flags: Pattern.CASE_INSENSITIVE | Pattern.MULTILINE
```

THREADS & MEMORY

```
Thread t = new Thread(runnable); t.start();
t.join();
Thread.ofVirtual().start(r);
synchronized (lock) { ... }        // mutual exclusion
volatile boolean flag;             // visibility, no atomicity
wait() / notifyAll()  inside synchronized, in a while-loop
```

States: NEW → RUNNABLE → (BLOCKED / WAITING / TIMED_WAITING) → TERMINATED. Establish **happens-before**; beware races & lock-ordering deadlock.

EXECUTORS & FUTURES

```
var ex = Executors.newFixedThreadPool(4);
Future<Integer> f = ex.submit(() -> 42);
ex.shutdown();

CompletableFuture.supplyAsync(this::load)
  .thenApply(this::parse)
  .thenCombine(other, (a,b) -> merge(a,b))
  .exceptionally(ex -> fallback);
ConcurrentHashMap  AtomicInteger  ReentrantLock  CountDownLatch
```

VIRTUAL THREADS

```
try (var ex =

Executors.newVirtualThreadPerTaskExecutor()) {
  IntStream.range(0, 10_000).forEach(i ->
    ex.submit(() -> blockingIo()));   // cheap; blocking OK
}
Thread.startVirtualThread(runnable);
```

Many virtual threads mount on few carrier (platform) threads; unmount while blocked. Prefer `ReentrantLock` over `synchronized` to avoid pinning.

PACKAGES & MODULES

```
package com.acme.app;  import java.util.*;
import static ...;

// module-info.java
module com.acme.app {
  requires java.sql;                // requires transitive / static
  exports com.acme.app.api;         // exports ... to m1, m2;
  opens com.acme.app.model;         // reflection access
  uses  com.acme.spi.Service;
  provides com.acme.spi.Service with com.acme.app.Impl;
}
```

JDK TOOLS & BUILD

```
javac java jar javadoc jshell jdeps jlink jpackage
jcmd <pid> Thread.print   jfr   −XX:StartFlightRecording
```

```
# Maven                       # Gradle
src/main/java  src/test/java  build.gradle(.kts)
mvn −q package                ./gradlew build
```

Maven = convention/XML; Gradle = programmable (Groovy/Kotlin DSL).

JUNIT 5

```
@Test void adds() { assertEquals(3, calc.add(1,2)); }
@BeforeEach @AfterEach @BeforeAll @AfterAll
assertThrows(X.class, () -> ...);
assertAll(...);
@DisplayName  @Nested  @Disabled  @Tag("slow")
@ParameterizedTest @ValueSource(ints={1,2,3})
@CsvSource({"1,2,3"})  @MethodSource("cases")
```

MOCKITO

```
@ExtendWith(MockitoExtension.class)
@Mock Repo repo;  @InjectMocks Service svc;

when(repo.find(1)).thenReturn(Optional.of(u));
when(repo.save(any())).thenThrow(new RuntimeException());
verify(repo, times(1)).save(eq(u));
var cap = ArgumentCaptor.forClass(User.class);
verify(repo).save(cap.capture());
spy(realObj);   // partial   −− don't mock types you don't own
```