Cheat Sheet OCP Oracle Certified Professional Java SE 21

Chapter 1 – Building Blocks

Packages and Imports

- If two classes have the same name (e.g., java.util.Date and java.sql.Date), you must qualify one explicitly; wildcard imports cause ambiguity.
- Wildcards (*) import all types from a package but not subpackages.
- Static imports are for static members only: import static java.util.Arrays.asList;
- Compiling with -d places .class files into a target folder; execution uses -cp or --class-path.

Data Types

- Primitives - defaults:
  boolean -> false; byte/short/int/long -> 0; float/double -> 0.0; char -> '\u0000'.
- short is signed, char is unsigned.
- Numeric literals: octal (prefix 0), hex (0x), binary (0b).
- Integer overflow silently wraps around.
- Integer.parseInt("123") returns int; Integer.valueOf("123") returns Integer object.

Variables and var

- Multiple variables may be declared in one statement only if same type.
- final arrays -> reference cannot change, but elements can.
- Local variables have no default values; must be initialized before use.
- var allowed only for local variables; type inferred from initializer.
- var must be initialized on the same line; cannot be reassigned to another type.
- var is not a reserved word; can still be used as an identifier.\

Garbage Collection (GC)

- The JVM automatically reclaims memory from objects **no longer reachable** by any live thread or static reference. Calling System.gc() is only a **suggestion**, not a guarantee.

Chapter 2 – Operators

- Precedence: unary -> multiplicative -> additive -> relational -> logical -> assignment.
- % keeps the sign of the left operand (2 % -5 = 2, -2 % 5 = -2).
- Arithmetic on smaller types (byte, short, char) promotes to int.
- Implicit casting only allowed from smaller to larger (widening). Narrowing needs explicit cast.
- == compares references for objects; .equals() compares values if overridden.
- instanceof returns false when reference is null.
- && and || short-circuit; & and | evaluate both sides (used also for bitwise ops).

Chapter 3 – Making Decisions

if Statements

- Condition must be boolean; numeric values are not auto-converted.
- Unreachable code after return, break, or continue is a compile-time error.

switch Expressions

- Supports both : (traditional) and -> (arrow) syntax. Arrow syntax implies no fall-through; colon requires explicit break.
- Cases must be compile-time constants or enums.

- Mixed case types (e.g., String and Integer) not allowed.
- Pattern matching for switch allows case Type t -> binds and tests simultaneously.
- More specific cases (subclasses) must come before general ones (superclasses).
- yield used in switch expressions to return a value without exiting the method.

Chapter 4 – Core APIs

Strings

- Immutable; operations produce new objects.
- String interning: literals and intern() share the same pool reference.
- StringBuilder is mutable; methods modify the same object.
- StringBuilder methods: append, delete, replace, reverse.

Arrays

- length is a field, not a method.
- binarySearch() returns negative insertion point if not found: -(index + 1).
- Arrays.compare() returns negative, zero, or positive.
- mismatch() returns first differing index or -1.

Math and Numbers

- ceil(3.14)=4.0, floor(3.14)=3.0.
- Math.random() gives double in [0.0, 1.0).
- For precision, use BigDecimal / BigInteger.

Dates and Times

- LocalDate, LocalTime, LocalDateTime, ZonedDateTime.
- LocalDate lacks time; ex: plusMinutes() -> invalid.
- Period.ofYears(1).ofWeeks(1) keeps only last call.
- Duration represents smaller units; cannot be used with LocalDate.

Chapter 5 – Methods

- All non-void methods must return a value in every path.
- Varargs (T...) must be last and appear only once.
- Access levels:
    - same class: all
    - same package: default, protected, public
    - subclass: protected, public
    - unrelated: public only
- static members shared across all instances.
- Overload resolution: Exact match -> Widening primitive -> Boxing -> Varargs
- int... and int[] cannot coexist as overloads.
- Static import allows access without class prefix, but only for members.

Chapter 6 – Class Design

Inheritance

- Transitive: if A extends B and B extends C, A is a subclass of C.
- A top-level class cannot be private or protected.

Constructors

- Must have same name as class, no return type.
- Default constructor is added only if no constructor exists.

- Must call super() or this() as first statement.

Initialization Order

- Static parent -> static child -> instance parent -> instance child -> constructors.

Overriding Rules

- Same signature and compatible return type (covariant allowed).
- Cannot reduce visibility. Cannot add new checked exceptions.
- final and static methods cannot be overridden (static methods are hidden).

Abstract & Immutable

- Abstract class cannot be instantiated; may include both abstract and concrete methods. Abstract methods cannot be final, private, or static.
- To design immutable classes: Class should be final or all constructors private. Fields private and final. No setters; return copies of mutable fields.

Chapter 7 – Beyond Classes

Interfaces

- All interface fields are implicitly public static final. Methods are implicitly public abstract unless default or static.
- Default methods can be overridden; static methods cannot.
- If two interfaces define same default method, implementing class must override it.

Enums

- Implicitly final and cannot extend other classes. Constructors are always private.
- Methods: values(), name(), ordinal().

Sealed Classes

- Restrict which classes can extend them via permits.
- Subclasses must declare final, sealed, or non-sealed.

Records

- Implicitly final, fields are private and final.
- Auto-generates canonical constructor, accessors, equals, hashCode, toString.
- Compact constructors allow validation but no parameter list.
- Cannot have instance initializers or non-static fields.

Nested and Inner Classes

- Inner (non-static) classes have reference to outer instance.
- Static nested classes behave like normal top-level classes.
- Local and anonymous classes can access only effectively final variables.

Chapter 8 – Lambdas and Functional Interfaces

- A functional interface has exactly one abstract method. May contain default, static, or private methods.
- Common built-ins:
  - Supplier<T> -> T get(); Consumer<T> -> void accept(T)
  - Predicate<T> -> boolean test(T); Function<T,R> -> R apply(T)
  - UnaryOperator<T> extends Function<T,T>; BinaryOperator<T> extends BiFunction<T,T,T>
- Method references: System.out::println, String::isEmpty, ClassName::new.
- Chaining: Predicates: and(), or(), negate(); Consumers: andThen(); Functions: compose(), andThen()

Chapter 9 – Collections and Generics

Collections

- List.of(), Set.of(), Map.of() -> immutable, throw exceptions on modification.
- Arrays.asList() -> fixed-size, modifiable elements.
- HashSet unordered, LinkedHashSet insertion order, TreeSet sorted.
- Maps: putIfAbsent(), replace(), getOrDefault(), entrySet(), keySet(), values()
- Comparator used for custom sort; Comparable defines natural order.
- SequencedCollection/SequencedMap (Java 21): getFirst(), getLast(), reversed().

Generics

- Wildcards:
  - ? – any type
  - ? extends T – read from, cannot write (producer)
  - ? super T – write to, cannot read (consumer)

Chapter 10 – Streams and Optionals

Streams

- Stream pipeline: source -> intermediate -> terminal.
- Intermediate operations are lazy (executed on terminal call).
- Intermediate ops: filter, map, flatMap, distinct, sorted, limit, skip.
- Terminal ops: forEach, collect, count, findFirst, reduce, allMatch, etc.
- After a terminal op, the stream is closed.
- reduce(identity, accumulator) combines stream elements.
- collect(Collectors.toList()), joining(), or custom collectors.
- Primitive streams: IntStream, LongStream, DoubleStream.
- Conversion: mapToInt(), mapToObj(), etc.
- Parallel streams: use parallelStream(). Order guaranteed only with forEachOrdered().

Optionals

- Optional.ofNullable(value) allows nulls.

Chapter 11 – Exceptions and Localization

Exception Hierarchy

- Throwable -> Error | Exception.
- RuntimeException and its subclasses are unchecked.
- Checked exceptions must be declared or caught.

Common Types

- Checked: IOException, SQLException, ParseException.
- Unchecked: NullPointerException, ClassCastException, IllegalArgumentException.
- Errors: StackOverflowError, OutOfMemoryError.

Rules

- Subclass overrides cannot add new checked exceptions.
- Multi-catch cannot combine parent-child exception types.
- finally executes always except when System.exit() or fatal error occurs.
- try-with-resources: closes in reverse order; resources must be AutoCloseable.

Localization

- Locale.getDefault(), Locale.US, etc.

- ResourceBundle hierarchy: language_country -> language -> default.
- MessageFormat.format() supports placeholders {0}, {1}.

Chapter 12 – Modules

- module-info.java defines module name, dependencies, and exports.
- exports makes packages visible to other modules.
- requires declares dependencies; requires transitive re-exports them.
- Module types:
  - Named: has module-info.java
  - Automatic: JAR on module path without descriptor
  - Unnamed: on classpath
- Built-in modules include java.base, java.sql, java.xml.
- Migration strategies:
  - Bottom-up: modularize lowest dependencies first.
  - Top-down: move entire project to module path, modularize entry points.

Chapter 13 – Concurrency

Threads

- Created via new Thread(Runnable), or Thread.ofVirtual() (Java 21).
- start() launches asynchronously; calling run() executes synchronously.
- Daemon threads don't prevent JVM shutdown.
- Lifecycle: new -> runnable -> blocked/waiting -> terminated.

Executor Framework

- ExecutorService for thread pooling; submit() returns Future.
- Always call shutdown() to avoid non-daemon threads hanging.
- ScheduledExecutorService for delayed or periodic tasks.

Synchronization

- synchronized locks object monitor; only one thread may enter block.
- Lock (e.g., ReentrantLock) offers more control: tryLock(timeout).
- Must always unlock in finally block.

Atomic and Concurrent Collections

- Atomic classes (AtomicInteger, etc.) provide lock-free thread-safe operations.
- Thread-safe collections: ConcurrentHashMap, ConcurrentLinkedQueue, ConcurrentSkipListMap/Set.
- Collections.synchronizedList() creates synchronized wrappers.

Common Issues

- Deadlock: threads waiting indefinitely on each other.
- Starvation: thread never scheduled.
- Livelock: threads active but making no progress.
- Race condition: concurrent unsynchronized modification causes inconsistent state.

Chapter 14 – I/O and Serialization

File and Path

- Path.of("..."), Paths.get("...") create path objects.
- resolve() concatenates paths; if second is absolute, it replaces first.
- relativize() requires both paths to be either relative or absolute.

- normalize() removes redundant . or ..
- Files.exists(), Files.copy(), Files.deleteIfExists() handle files safely.
- Files.copy() throws if target exists unless using REPLACE_EXISTING.
- Symbolic links considered equal with Files.isSameFile().

Streams

- Byte streams: InputStream/OutputStream.
- Character streams: Reader/Writer.
- Buffered streams improve performance.
- Files.newBufferedReader(path) and Files.newBufferedWriter(path) wrap file IO.
- Low-level: FileInputStream, FileReader; High-level: BufferedReader, PrintWriter.

Serialization

- Class must implement Serializable.
- Non-serializable fields must be transient.
- All object graph members must be serializable or null, else NotSerializableException.
- Writing: ObjectOutputStream.writeObject(obj).
- Reading: ObjectInputStream.readObject().

Console

- System.console() may return null in some environments (IDE).
- Closing System.out or System.in disables further use.