Apache Lucene Cheat Sheet

Lucene 10.x · requires Java 21 · lucene.apache.org/core/

Embedded, in-process Java search library — no daemon, no network protocol. You own concurrency, persistence, replication and sharding. Solr, Elasticsearch and OpenSearch are all built on it. One skeleton per feature area; see the online reference for full detail.

BUILD COORDINATES

```
<!-- Maven -->
<dependency>
  <groupId>org.apache.lucene</groupId>
  <artifactId>lucene-core</artifactId>
  <version>10.x</version>
</dependency>
// Gradle
implementation "org.apache.lucene:lucene-core:10.x"
```

Companion modules: `lucene-analysis-common`, `-queryparser`, `-facet`, `-highlighter`, `-suggest`, `-join`, `-grouping`, `-expressions`, `-queries`, `-spatial-extras`, `-monitor`, `-backward-codecs`, `-test-framework`.

INDEX + SEARCH ROUND-TRIP

```
Analyzer a = new StandardAnalyzer();
Directory dir = FSDirectory.open(Path.of("idx"));

try (IndexWriter w = new IndexWriter(
        dir, new IndexWriterConfig(a))) {
  Document d = new Document();
  d.add(new StringField("id","1",Store.YES));
  d.add(new TextField("body",text,Store.NO));
  w.addDocument(d);            // or
updateDocument(term,d)
}

try (DirectoryReader r = DirectoryReader.open(dir)) {
  IndexSearcher s = new IndexSearcher(r);
  Query q = new TermQuery(new Term("body","lucene"));
  TopDocs top = s.search(q, 10);
  for (ScoreDoc h : top.scoreDocs) {
    Document doc = s.storedFields().document(h.doc);
  }
}
```

FIELD-TYPE CAPABILITY MATRIX

Field	search	store	sort/facet	range	vector
TextField	✓	opt	·	·	·
StringField / KeywordField	✓	opt	·	·	·
StoredField	·	✓	·	·	·
IntPoint / LongPoint / DoublePoint	✓	·	·	✓	·
*DocValuesField	·	·	✓	skip	·
IntField / LongField (point+DV)	✓	·	✓	✓	·
KnnFloatVectorField / KnnByteVectorField	·	·	·	·	✓

`Document` is an ordered list of `Fields` — no schema. Combine types on one logical field to get several capabilities. `FieldType` / `IndexOptions` pick docs / freqs / positions / offsets.

ANALYSIS CHAIN

```
Analyzer a = CustomAnalyzer.builder()
  .withTokenizer("standard")
  .addCharFilter("htmlstrip")
  .addTokenFilter("lowercase")
  .addTokenFilter("asciiFolding")
  .addTokenFilter("stop")
  .addTokenFilter("synonymGraph",
      "synonyms","syn.txt","expand","true")
  .addTokenFilter("flattenGraph")
  .addTokenFilter("porterStem")
  .build();
```

Pipeline: `CharFilter` → `Tokenizer` → `TokenFilter`*. Stream lifecycle: `reset()` → `incrementToken()` → `end()` → `close()`. `PerFieldAnalyzerWrapper` for per-field rules.

INDEXWRITERCONFIG KNOBS

```
IndexWriterConfig c = new IndexWriterConfig(a);
c.setOpenMode(OpenMode.CREATE_OR_APPEND);
c.setRAMBufferSizeMB(256);          // flush trigger
c.setUseCompoundFile(true);
c.setSimilarity(new BM25Similarity());
c.setIndexSort(new Sort(
    new SortField("price", Type.LONG)));
c.setMergePolicy(new TieredMergePolicy());   //
default
c.setMergeScheduler(new ConcurrentMergeScheduler());
c.setSoftDeletesField("__soft_deletes");
```

Writes: `addDocument`, `updateDocument` (atomic delete-by-Term + add), `softUpdateDocument`, `deleteDocuments(Query|Term)`. Durability: `commit` vs `flush`; `prepareCommit` two-phase; `rollback`. `TieredMergePolicy` knobs: `maxMergedSegmentMB`, `segmentsPerTier`, `deletesPctAllowed`.

CORE QUERY BUILDERS

```
Query bq = new BooleanQuery.Builder()
  .add(new TermQuery(new Term("title","lucene")),
Occur.MUST)
  .add(new TermQuery(new Term("tag","search")),
Occur.FILTER)
  .add(new TermQuery(new Term("tag","legacy")),
Occur.MUST_NOT)
  .setMinimumNumberShouldMatch(1)
  .build();                          // ≤ 1024 clauses

Query pq = new PhraseQuery.Builder()
  .add(new Term("body","apache"))
  .add(new Term("body","lucene")).setSlop(1).build();

Query rq = LongPoint.newRangeQuery("price", 10L,
100L);
Query rr = new IndexOrDocValuesQuery(rq,
dvRangeQuery);
Query fz = new FuzzyQuery(new Term("name","lucne"),
2);
```

Also `PrefixQuery`, `WildcardQuery`, `RegexpQuery`, `TermRangeQuery`, `TermInSetQuery`, `ConstantScoreQuery`, `BoostQuery`, `DisjunctionMaxQuery`. Parsers: `QueryParser` (analyzed, can throw), `SimpleQueryParser` (never throws), `StandardQueryParser`. Proximity: `Intervals.ordered` / `unordered` / `maxgaps` / `containing` → `IntervalQuery`.

BM25 + EXPLANATION

```
searcher.setSimilarity(new BM25Similarity(1.2f,
0.75f)); // k1,b
Explanation e = searcher.explain(query, docId);
System.out.println(e);   // per-term tf, df, idf,
fieldLen
```

Default since Lucene 6. Alternatives: `BooleanSimilarity`, `ClassicSimilarity`, `DFR/IB`, `LMDirichlet`, `LMJelinekMercer`; `PerFieldSimilarityWrapper`. Index-time similarity sets norms, query-time sets ranking.

FUNCTION SCORING, SORT & SEARCHAFTER

```
DoubleValuesSource pop =
DoubleValuesSource.fromLongField("pop");
Query fs = new FunctionScoreQuery(base,
    DoubleValuesSource.SCORES.multiply(pop));
Query bv = FunctionScoreQuery.boostByValue(base,
pop);
Query fx =
FeatureField.newSaturationQuery("features","pagerank"
);

Sort sort = new Sort(
    new SortField("price", Type.LONG, false),
    SortField.FIELD_SCORE);
TopDocs p1 = searcher.search(q, 20, sort);
FieldDoc last = (FieldDoc)
p1.scoreDocs[p1.scoreDocs.length-1];
TopDocs p2 = searcher.searchAfter(last, q, 20, sort);
```

`lucene-expressions`: `JavascriptCompiler.compile("ln(pop)+_score")` → `DoubleValuesSource` for score / sort / facet.

COLLECTORMANAGER + CONCURRENT SEARCH

```
IndexSearcher s = new IndexSearcher(reader,
executor); // pool
// intra-segment concurrency: 10.x

var cm = new TopScoreDocCollectorManager(
    10, null, Integer.MAX_VALUE); //
totalHitsThreshold
TopDocs top = s.search(query, cm);

var fcm = new TopFieldCollectorManager(sort, 10,
1000);
var hits = s.search(query, new
TotalHitCountCollectorManager());
```

`TotalHits.Relation`: `EQUAL_TO` vs `GREATER_THAN_OR_EQUAL_TO` once the threshold is passed. Early termination via the threshold or `TimeLimitingCollector`. Custom: implement `Collector` + `CollectorManager` (reduce merges per-slice top-N via `TopDocs.merge`).

FACETS: TAXONOMY VS. SSDV

```
FacetsConfig cfg = new FacetsConfig();
cfg.setHierarchical("path", true);

// (A) taxonomy sidecar index
var tw = new DirectoryTaxonomyWriter(taxoDir);
doc.add(new FacetField("author","kafka"));
w.addDocument(cfg.build(tw, doc));
Facets f = new FastTaxonomyFacetCounts(taxoReader,
cfg, fc);

// (B) SortedSetDocValues -- no sidecar
doc.add(new
SortedSetDocValuesFacetField("author","kafka"));
Facets g = new SortedSetDocValuesFacetCounts(state,
fc);

FacetResult r = f.getTopChildren(10, "author");
```

Drill-down / sideways: `DrillDownQuery`, `DrillSideways`. Range facets: `LongRangeFacetCounts`.

KNN / HNSW VECTOR SEARCH

```
doc.add(new KnnFloatVectorField("vec", embedding,
    VectorSimilarityFunction.COSINE));    //
DOT_PRODUCT | EUCLIDEAN | MIP

Query knn = new KnnFloatVectorQuery("vec", queryVec,
    100, filterQuery);          // k, optional pre-
filter
// hybrid: BooleanQuery(SHOULD lexical, SHOULD knn)
or rescore
TopDocs hits = searcher.search(knn, 100);
```

HNSW lives in the `KnnVectorsFormat` (`M`, `beamWidth`); scalar `int8`/`int4` and binary quantisation formats. Panama Vector API accelerates distance maths.

NEAR-REAL-TIME (SEARCHERMANAGER) LOOP

```
SearcherManager mgr = new SearcherManager(writer, new
SearcherFactory());

var reopen = new ControlledRealTimeReopenThread<>(
    writer, mgr, 5.0, 0.025);   // max / min stale
seconds
reopen.start();

// per request:
IndexSearcher s = mgr.acquire();
try { s.search(q, 10); } finally { mgr.release(s); }

// after writes: writer.addDocument(...);
mgr.maybeRefresh();
```

Never `close()` a reader other threads may hold — rely on `incRef`/`decRef`. `SearcherLifetimeManager` pins a searcher for stable deep paging.

OPS ONE-LINERS

```
// compact a STATIC index only (expensive, rewrites
all segments)
writer.forceMerge(1);
writer.forceMergeDeletes();

// verify integrity / list segments
java -cp lucene-core.jar
org.apache.lucene.index.CheckIndex /path/idx
// programmatic: new CheckIndex(dir).checkIndex()

// upgrade segments one major version forward
java -cp "lucene-core.jar:lucene-backward-codecs.jar"
\
  org.apache.lucene.index.IndexUpgrader /path/idx
```

Read-compat is "one major version back" via `lucene-backward-codecs`. Browse an index with **Luke**. Benchmark with **luceneutil** / the nightly benchmarks.

LUCENE VS. SOLR VS. ELASTICSEARCH VS. OPENSEARCH

Pick	When
Lucene	Single JVM, lowest latency, full API control, smallest footprint. You build sharding, replication, failover, a query protocol, security, snapshots.
Solr	Want a search server on Lucene with mature faceting / dismax, schema & config files, SolrCloud + ZooKeeper. Apache-2.0.
Elasticsearch	Want batteries-included clustering / ILM / ingest / security / Kibana. SSPL / Elastic License v2.
OpenSearch	Want the ES 7.10 feature set fully Apache-2.0, AWS-led, OpenSearch Dashboards.

Solr, Elasticsearch and OpenSearch all embed Lucene — every concept here transfers one layer up.