# Core

## Information Retrieval:

* Higher Package:
  1. org.apache.lucene.index: Code to maintain and access indices.
     + IR Class:

1. IndexReader: IndexReader is an abstract class, providing an interface for accessing a point-in-time view of an index.
2. IndexReader.CacheKey: A cache key identifying a resource that is being cached on.
3. IndexReaderContext: A struct like class that represents a hierarchical relationship between IndexReader instances.
4. IndexWriter: An IndexWriter creates and maintains an index.
5. IndexWriterConfig: Holds all the configuration that is used to create an IndexWriter.
6. LeafMetaData: Provides read-only metadata about a leaf.
7. LeafReader: LeafReader is an abstract class, providing an interface for accessing an index.
8. LeafReaderContext: IndexReaderContext for LeafReader instances.
9. LiveIndexWriterConfig: Holds all the configuration used by IndexWriter with few setters for settings that can be changed on an IndexWriter instance "live".
10. MultiFields : Provides a single Fields term index view over an IndexReader.
    * + Algorithm Class:
      + Data Structure Class:
        1. IndexableField: Represents a single field for indexing.
        2. IndexableFieldType: Describes the properties of a field.
        3. IndexFileNames: This class contains useful constants representing filenames and extensions used by lucene, as well as convenience methods for querying whether a file name matches an extension (matchesExtension), as well as generating file names from a segment name, generation and extension ( fileNameFromGeneration, segmentFileName).
        4. Term : A Term represents a word from text.
        5. TermContext : Maintains a IndexReader TermState view over IndexReader instances containing a single term.
        6. Terms : Access to the terms in a specific field.
        7. TermsEnum : Iterator to seek (TermsEnum.seekCeil(BytesRef), TermsEnum.seekExact(BytesRef)) or step through (BytesRefIterator.next() terms to obtain frequency information (TermsEnum.docFreq()), PostingsEnum or PostingsEnum for the current term (TermsEnum.postings(org.apache.lucene.index.PostingsEnum).
        8. TermState : Encapsulates all required internal state to position the associated TermsEnum without re:seeking.
      + Low-Level Class:
        1. IndexReader.CacheHelper: A utility class that gives hooks in order to help build a cache based on the data that is contained in this index.
        2. IndexReader.ClosedListener: A listener that is called when a resource gets closed.
        3. IndexWriter.IndexReaderWarmer: If DirectoryReader.open(IndexWriter) has been called (ie, this writer is in near real-time mode), then after a merge completes, this class can be invoked to warm the reader on the newly merged segment, before the merge commits.
        4. MergePolicy.MergeContext: This interface represents the current context of the merge selection process.
        5. PointValues.IntersectVisitor: We recurse the BKD tree, using a provided instance of this to guide the recursion.
        6. QueryTimeout: Base for query timeout implementations, which will provide a shouldExit() method, used with ExitableDirectoryReader.
        7. TwoPhaseCommit: An interface for implementations that support 2-phase commit.
        8. BaseCompositeReader<R extends IndexReader>: Base class for implementing CompositeReaders based on an array of sub-readers.
        9. BinaryDocValues: A per-document numeric value.
        10. CheckIndex: Basic tool and API to check the health of an index and write a new segments file that removes reference to problematic segments.
        11. CheckIndex.Options: Run-time configuration options for CheckIndex commands.
        12. CheckIndex.Status: Returned from CheckIndex.checkIndex() detailing the health and status of the index.
        13. CheckIndex.Status.DocValuesStatus: Status from testing DocValues
        14. CheckIndex.Status.FieldInfoStatus: Status from testing field infos.
        15. CheckIndex.Status.FieldNormStatus: Status from testing field norms.
        16. CheckIndex.Status.IndexSortStatus: Status from testing index sort
        17. CheckIndex.Status.LiveDocStatus: Status from testing livedocs
        18. CheckIndex.Status.PointsStatus: Status from testing PointValues
        19. CheckIndex.Status.SegmentInfoStatus: Holds the status of each segment in the index.
        20. CheckIndex.Status.StoredFieldStatus: Status from testing stored fields.
        21. CheckIndex.Status.TermIndexStatus: Status from testing term index.
        22. CheckIndex.Status.TermVectorStatus: Status from testing stored fields.
        23. CheckIndex.VerifyPointsVisitor: Walks the entire N-dimensional points space, verifying that all points fall within the last cell's boundaries.
        24. CodecReader: LeafReader implemented by codec APIs.
        25. CompositeReader: Instances of this reader type can only be used to get stored fields from the underlying LeafReaders, but it is not possible to directly retrieve postings.
        26. CompositeReaderContext: IndexReaderContext for CompositeReader instance.
        27. ConcurrentMergeScheduler: A MergeScheduler that runs each merge using a separate thread.
        28. DirectoryReader: DirectoryReader is an implementation of CompositeReader that can read indexes in a Directory.
        29. DocIDMerger<T extends DocIDMerger.Sub>: Utility class to help merging documents from sub-readers according to either simple concatenated (unsorted) order, or by a specified index-time sort, skipping deleted documents and remapping non-deleted documents.
        30. DocIDMerger.Sub: Represents one sub-reader being merged
        31. DocValues: This class contains utility methods and constants for DocValues
        32. EmptyDocValuesProducer: Abstrast base class implementing a DocValuesProducer that has no doc values.
        33. ExitableDirectoryReader: The ExitableDirectoryReader wraps a real index DirectoryReader and allows for a QueryTimeout implementation object to be checked periodically to see if the thread should exit or not.
        34. ExitableDirectoryReader.ExitableFilterAtomicReader: Wrapper class for another FilterAtomicReader.
        35. ExitableDirectoryReader.ExitableSubReaderWrapper: Wrapper class for a SubReaderWrapper that is used by the ExitableDirectoryReader.
        36. ExitableDirectoryReader.ExitableTerms: Wrapper class for another Terms implementation that is used by ExitableFields.
        37. ExitableDirectoryReader.ExitableTermsEnum: Wrapper class for TermsEnum that is used by ExitableTerms for implementing an exitable enumeration of terms.
        38. FieldInfo: Access to the Field Info file that describes document fields and whether or not they are indexed.
        39. FieldInfos: Collection of FieldInfos (accessible by number or by name).
        40. FieldInvertState: This class tracks the number and position / offset parameters of terms being added to the index.
        41. Fields: Provides a Terms index for fields that have it, and lists which fields do.
        42. FilterBinaryDocValues: Delegates all methods to a wrapped BinaryDocValues.
        43. FilterCodecReader: A FilterCodecReader contains another CodecReader, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.
        44. FilterDirectoryReader: A FilterDirectoryReader wraps another DirectoryReader, allowing implementations to transform or extend it.
        45. FilterDirectoryReader.SubReaderWrapper: Factory class passed to FilterDirectoryReader constructor that allows subclasses to wrap the filtered DirectoryReader's subreaders.
        46. FilteredTermsEnum: Abstract class for enumerating a subset of all terms.
        47. FilterLeafReader: A FilterLeafReader contains another LeafReader, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.
        48. FilterLeafReader.FilterFields: Base class for filtering Fields implementations.
        49. FilterLeafReader.FilterPostingsEnum: Base class for filtering PostingsEnum implementations.
        50. FilterLeafReader.FilterTerms: Base class for filtering Terms implementations.
        51. FilterLeafReader.FilterTermsEnum: Base class for filtering TermsEnum implementations.
        52. FilterMergePolicy: A wrapper for MergePolicy instances.
        53. FilterNumericDocValues: Delegates all methods to a wrapped NumericDocValues.
        54. IndexCommit: Expert: represents a single commit into an index as seen by the IndexDeletionPolicy or IndexReader.
        55. IndexDeletionPolicy: Expert: policy for deletion of stale index commits.
        56. IndexUpgrader: This is an easy-to-use tool that upgrades all segments of an index from previous Lucene versions to the current segment file format.
        57. KeepOnlyLastCommitDeletionPolicy: This IndexDeletionPolicy implementation that keeps only the most recent commit and immediately removes all prior commits after a new commit is done.
        58. LegacyBinaryDocValues: Deprecated Use BinaryDocValues instead.
        59. LegacyBinaryDocValuesWrapper: Deprecated Implement BinaryDocValues directly.
        60. LegacyNumericDocValues: Deprecated Use NumericDocValues instead.
        61. LegacyNumericDocValuesWrapper: Deprecated Implement NumericDocValues directly.
        62. LegacySortedDocValues: Deprecated Use SortedDocValues instead.
        63. LegacySortedDocValuesWrapper: Deprecated Implement SortedDocValues directly.
        64. LegacySortedNumericDocValues: Deprecated Use SortedNumericDocValues instead.
        65. LegacySortedNumericDocValuesWrapper: Deprecated Implement SortedNumericDocValues directly.
        66. LegacySortedSetDocValues: Deprecated Use SortedSetDocValues instead.
        67. LegacySortedSetDocValuesWrapper: Deprecated Implement SortedSetDocValues directly.
        68. LogByteSizeMergePolicy: This is a LogMergePolicy that measures size of a segment as the total byte size of the segment's files.
        69. LogDocMergePolicy: This is a LogMergePolicy that measures size of a segment as the number of documents (not taking deletions into account).
        70. LogMergePolicy: This class implements a MergePolicy that tries to merge segments into levels of exponentially increasing size, where each level has fewer segments than the value of the merge factor.
        71. MappedMultiFields: A Fields implementation that merges multiple Fields into one, and maps around deleted documents.
        72. MergePolicy: Expert: a MergePolicy determines the sequence of primitive merge operations.
        73. MergePolicy.MergeSpecification: A MergeSpecification instance provides the information necessary to perform multiple merges.
        74. MergePolicy.OneMerge: OneMerge provides the information necessary to perform an individual primitive merge operation, resulting in a single new segment.
        75. MergePolicy.OneMergeProgress: Progress and state for an executing merge.
        76. MergePolicyWrapper: Deprecated replace with FilterMergePolicy
        77. MergeRateLimiter : This is the RateLimiter that IndexWriter assigns to each running merge, to give MergeSchedulers ionice like control.
        78. MergeScheduler : Expert: IndexWriter uses an instance implementing this interface to execute the merges selected by a MergePolicy.
        79. MergeState : Holds common state used during segment merging.
        80. MergeState.DocMap: A map of doc IDs.
        81. MultiDocValues : A wrapper for CompositeIndexReader providing access to DocValues.
        82. MultiDocValues.MultiSortedDocValues: Implements SortedDocValues over n subs, using an OrdinalMap
        83. MultiDocValues.MultiSortedSetDocValues: Implements MultiSortedSetDocValues over n subs, using an OrdinalMap
        84. MultiPostingsEnum : Exposes PostingsEnum, merged from PostingsEnum API of sub:segments.
        85. MultiPostingsEnum.EnumWithSlice: Holds a PostingsEnum along with the corresponding ReaderSlice.
        86. MultiReader : A CompositeReader which reads multiple indexes, appending their content.
        87. MultiTerms : Exposes flex API, merged from flex API of sub:segments.
        88. MultiTermsEnum : Exposes TermsEnum API, merged from TermsEnum API of sub:segments.
        89. NoDeletionPolicy : An IndexDeletionPolicy which keeps all index commits around, never deleting them.
        90. NoMergePolicy : A MergePolicy which never returns merges to execute.
        91. NoMergeScheduler : A MergeScheduler which never executes any merges.
        92. NumericDocValues : A per:document numeric value.
        93. OneMergeWrappingMergePolicy : A wrapping merge policy that wraps the MergePolicy.OneMerge objects returned by the wrapped merge policy.
        94. OrdinalMap : Maps per:segment ordinals to/from global ordinal space, using a compact packed:ints representation.
        95. OrdTermState : An ordinal based TermState
        96. ParallelCompositeReader : An CompositeReader which reads multiple, parallel indexes.
        97. ParallelLeafReader : An LeafReader which reads multiple, parallel indexes.
        98. PersistentSnapshotDeletionPolicy : A SnapshotDeletionPolicy which adds a persistence layer so that snapshots can be maintained across the life of an application.
        99. PointValues : Access to indexed numeric values.
        100. PostingsEnum : Iterates through the postings.
        101. PrefixCodedTerms : Prefix codes term instances (prefixes are shared).
        102. PrefixCodedTerms.Builder: Builds a PrefixCodedTerms: call add repeatedly, then finish.
        103. PrefixCodedTerms.TermIterator: An iterator over the list of terms stored in a PrefixCodedTerms.
        104. QueryTimeoutImpl : An implementation of QueryTimeout that can be used by the ExitableDirectoryReader class to time out and exit out when a query takes a long time to rewrite.
        105. ReaderManager : Utility class to safely share DirectoryReader instances across multiple threads, while periodically reopening.
        106. ReaderSlice : Subreader slice from a parent composite reader.
        107. ReaderUtil : Common util methods for dealing with IndexReaders and IndexReaderContexts.
        108. SegmentCommitInfo : Embeds a [read:only] SegmentInfo and adds per:commit fields.
        109. SegmentInfo : Information about a segment such as its name, directory, and files related to the segment.
        110. SegmentInfos : A collection of segmentInfo objects with methods for operating on those segments in relation to the file system.
        111. SegmentInfos.FindSegmentsFile<T>: Utility class for executing code that needs to do something with the current segments file.
        112. SegmentReader : IndexReader implementation over a single segment.
        113. SegmentReadState : Holder class for common parameters used during read.
        114. SegmentWriteState : Holder class for common parameters used during write.
        115. SerialMergeScheduler : A MergeScheduler that simply does each merge sequentially, using the current thread.
        116. SimpleMergedSegmentWarmer : A very simple merged segment warmer that just ensures data structures are initialized.
        117. SingleTermsEnum : Subclass of FilteredTermsEnum for enumerating a single term.
        118. SlowCodecReaderWrapper : Wraps arbitrary readers for merging.
        119. SnapshotDeletionPolicy : An IndexDeletionPolicy that wraps any other IndexDeletionPolicy and adds the ability to hold and later release snapshots of an index.
        120. SoftDeletesDirectoryReaderWrapper : This reader filters out documents that have a doc values value in the given field and treat these documents as soft deleted.
        121. SoftDeletesRetentionMergePolicy : This MergePolicy allows to carry over soft deleted documents across merges.
        122. SortedDocValues : A per:document byte[] with presorted values.
        123. SortedNumericDocValues : A list of per:document numeric values, sorted according to Long.compare(long, long).
        124. SortedSetDocValues : A multi:valued version of SortedDocValues.
        125. StandardDirectoryReader : Default implementation of DirectoryReader.
        126. StoredFieldVisitor : Expert: provides a low:level means of accessing the stored field values in an index.
        127. TieredMergePolicy : Merges segments of approximately equal size, subject to an allowed number of segments per tier.
        128. MergeScore : Holds score and explanation for a single candidate merge.
        129. TwoPhaseCommitTool : A utility for executing 2:phase commit on several objects.
        130. UpgradeIndexMergePolicy : This MergePolicy is used for upgrading all existing segments of an index when calling IndexWriter.forceMerge(int).
      + Other Class:
        1. AutomatonTermsEnum: A FilteredTermsEnum that enumerates terms based upon what is accepted by a DFA.
      + Enum Classes(Name represents):
        1. DocValuesType : DocValues types.
        2. AcceptStatus : Return value, if term should be accepted or the iteration should END.
        3. IndexOptions : Controls how much information is stored in the postings lists.
        4. IndexWriterConfig.OpenMode: Specifies the open mode for IndexWriter.
        5. MergePolicy.OneMergeProgress.PauseReason: Reason for pausing the merge thread.
        6. MergeTrigger: MergeTrigger is passed to MergePolicy.findMerges(MergeTrigger, SegmentInfos, MergePolicy.MergeContext) to indicate the event that triggered the merge.
        7. PointValues.Relation: Used by PointValues.intersect (org.apache.lucene.index.PointValues.IntersectVisitor) to check how each recursive cell corresponds to the query.
        8. StoredFieldVisitor.Status: Enumeration of possible return values for StoredFieldVisitor.needsField(org.apache.lucene.index.FieldInfo).
        9. TermsEnum.SeekStatus: Represents returned result from TermsEnum.seekCeil(org.apache.lucene.util.BytesRef).
      + Exception Classes:
        1. CorruptIndexException: This exception is thrown when Lucene detects an inconsistency in the index.
        2. ExitableDirectoryReader.ExitingReaderException: Exception that is thrown to prematurely terminate a term enumeration.
        3. IndexFormatTooNewException: This exception is thrown when Lucene detects an index that is newer than this Lucene version.
        4. IndexFormatTooOldException: This exception is thrown when Lucene detects an index that is too old for this Lucene version
        5. IndexNotFoundException: Signals that no index was found in the Directory.
        6. MergePolicy.MergeAbortedException: Thrown when a merge was explicitly aborted because IndexWriter.abortMerges() was called.
        7. MergePolicy.MergeException: Exception thrown if there are any problems while executing a merge.
        8. TwoPhaseCommitTool.CommitFailException: Thrown by TwoPhaseCommitTool.execute(TwoPhaseCommit...) when an object fails to commit().
        9. TwoPhaseCommitTool.PrepareCommitFailException: Thrown by TwoPhaseCommitTool.execute(TwoPhaseCommit...) when an object fails to prepareCommit().

* Lower Package:
  1. org.apache.lucene.codecs: Codecs API: API for customization of the encoding and structure of the index.
     1. BlockTermState: Holds all state required for PostingsReaderBase to produce a PostingsEnum without re-seeking the terms dict.
     2. CodecUtil: Utility class for reading and writing versioned headers.
     3. CompoundFormat: Encodes/decodes compound files
     4. DocValuesConsumer: Abstract API that consumes numeric, binary and sorted docvalues.
     5. DocValuesFormat: Encodes/decodes per-document values.
     6. DocValuesProducer: Abstract API that produces numeric, binary, sorted, sortedset, and sortednumeric docvalues.
     7. FieldInfosFormat: Encodes/decodes FieldInfos
     8. FieldsConsumer: Abstract API that consumes terms, doc, freq, prox, offset and payloads postings.
     9. FieldsProducer: Abstract API that produces terms, doc, freq, prox, offset and payloads postings.
     10. FilterCodec: A codec that forwards all its method calls to another codec.
     11. LegacyDocValuesIterables: Bridge helper methods for legacy codecs to map sorted doc values to iterables.
     12. LiveDocsFormat: Format for live/deleted documents
     13. MultiLevelSkipListReader: This abstract class reads skip lists with multiple levels.
     14. MultiLevelSkipListWriter: This abstract class writes skip lists with multiple levels.
     15. MutablePointValues: PointValues whose order of points can be changed.
     16. NormsConsumer: Abstract API that consumes normalization values.
     17. NormsFormat: Encodes/decodes per-document score normalization values.
     18. NormsProducer: Abstract API that produces field normalization values
     19. PointsFormat: Encodes/decodes indexed points.
     20. PointsReader: Abstract API to visit point values.
     21. PointsWriter: Abstract API to write points
     22. PostingsFormat: Encodes/decodes terms, postings, and proximity data.
     23. PostingsReaderBase: The core terms dictionaries (BlockTermsReader, BlockTreeTermsReader) interact with a single instance of this class to manage creation of PostingsEnum and PostingsEnum instances.
     24. PostingsWriterBase: Class that plugs into term dictionaries, such as BlockTreeTermsWriter, and handles writing postings.
     25. PushPostingsWriterBase: Extension of PostingsWriterBase, adding a push API for writing each element of the postings.
     26. SegmentInfoFormat: Expert: Controls the format of the SegmentInfo (segment metadata file).
     27. StoredFieldsFormat: Controls the format of stored fields
     28. StoredFieldsReader: Codec API for reading stored fields.
     29. StoredFieldsWriter: Codec API for writing stored fields: For every document, StoredFieldsWriter.startDocument() is called, informing the Codec that a new document has started.
     30. TermStats: Holder for per-term statistics.
     31. TermVectorsFormat: Controls the format of term vectors
     32. TermVectorsReader: Codec API for reading term vectors:
     33. TermVectorsWriter: Codec API for writing term vectors: For every document, TermVectorsWriter.startDocument(int) is called, informing the Codec how many fields will be written.
  2. org.apache.lucene.codecs.blocktree: BlockTree terms dictionary. (stores the prefix trie in memory as an FST as the index structure.)

1. BlockTreeTermsReader: A block-based terms index and dictionary that assigns terms to variable length blocks according to how they share prefixes.
2. BlockTreeTermsWriter: Block-based terms index and dictionary writer.
3. FieldReader: BlockTree's implementation of Terms.
4. Stats: BlockTree statistics for a single field returned by FieldReader.getStats().
   1. org.apache.lucene.codecs.compressing: StoredFieldsFormat that allows cross-document and cross-field compression of stored fields.
   2. org.apache.lucene.codecs.lucene50: Components from the Lucene 5.0 index format See org.apache.lucene.codecs.lucene50 for an overview of the index format.
   3. org.apache.lucene.codecs.lucene60: Components from the Lucene 6.0 index format.
   4. org.apache.lucene.codecs.lucene62: Components from the Lucene 6.2 index format See org.apache.lucene.codecs.lucene70 for an overview of the current index format.
   5. org.apache.lucene.codecs.lucene70: Lucene 7.0 file format. (Define File index formats)
   6. org.apache.lucene.codecs.perfield: Postings format that can delegate to different formats per-field.
   7. org.apache.lucene.document: The logical representation of a Document for indexing and searching.
      1. BinaryDocValuesField: Field that stores a per-document BytesRef value.
      2. BinaryPoint: An indexed binary field for fast range filters.
      3. DateTools: Provides support for converting dates to strings and vice-versa.
      4. Document: Documents are the unit of indexing and search.
      5. DocumentStoredFieldVisitor: A StoredFieldVisitor that creates a Document from stored fields.
      6. DoubleDocValuesField: Syntactic sugar for encoding doubles as NumericDocValues via Double.doubleToRawLongBits(double).
      7. DoublePoint: An indexed double field for fast range filters.
      8. DoubleRange: An indexed Double Range field.
      9. FeatureField: Field that can be used to store static scoring factors into documents.
      10. Field: Expert: directly create a field for a document.
      11. FieldType: Describes the properties of a field.
      12. FloatDocValuesField: Syntactic sugar for encoding floats as NumericDocValues via Float.floatToRawIntBits(float).
      13. FloatPoint: An indexed float field for fast range filters.
      14. FloatRange: An indexed Float Range field.
      15. IntPoint: An indexed int field for fast range filters.
      16. IntRange: An indexed Integer Range field.
      17. LongPoint: An indexed long field for fast range filters.
      18. LongRange: An indexed Long Range field.
      19. NumericDocValuesField: Field that stores a per-document long value for scoring, sorting or value retrieval.
      20. SortedDocValuesField: Field that stores a per-document BytesRef value, indexed for sorting.
      21. SortedNumericDocValuesField: Field that stores a per-document long values for scoring, sorting or value retrieval.
      22. SortedSetDocValuesField: Field that stores a set of per-document BytesRef values, indexed for faceting,grouping,joining.
      23. StoredField: A field whose value is stored so that IndexSearcher.doc(int) and IndexReader.document() will return the field and its value.
      24. StringField: A field that is indexed but not tokenized: the entire String value is indexed as a single token.
      25. TextField: A field that is indexed and tokenized, without term vectors.
   8. org.apache.lucene.search: Code to search indices.
   9. org.apache.lucene.search.similarities: This package contains the various ranking models that can be used in Lucene.
   10. org.apache.lucene.search.spans: The calculus of spans.
   11. org.apache.lucene.store: Binary i/o API, used for all index data.
   12. org.apache.lucene.util: Some utility classes.
   13. org.apache.lucene.util.automaton: Finite-state automaton for regular expressions.
   14. org.apache.lucene.util.bkd: Block KD-tree, implementing the generic spatial data structure described in this paper.
   15. org.apache.lucene.util.fst: Finite state transducers
   16. org.apache.lucene.util.graph: Utility classes for working with token streams as graphs.
   17. org.apache.lucene.util.mutable: Comparable object wrappers
   18. org.apache.lucene.util.packed: Packed integer arrays and streams

## Natural Language Processing:

1. org.apache.lucene.analysis: Text analysis
   * IR Related Classes:
     1. CachingTokenFilter: This class can be used if the token attributes of a TokenStream are intended to be consumed more than once.
     2. LowerCaseFilter: Normalizes token text to lower case.
     3. StopFilter: Removes stop words from a token stream.
     4. WordlistLoader: Loader for text files that represent a list of stopwords.
     5. Analyzer: An Analyzer builds TokenStreams, which analyze text.
     6. FilteringTokenFilter: Abstract base class for TokenFilters that may remove tokens.
     7. TokenFilter: A TokenFilter is a TokenStream whose input is another TokenStream.
     8. Tokenizer: A Tokenizer is a TokenStream whose input is a Reader.
   * Algorithm Classes:
   * Data Structures Classes:
     1. TokenStream: A TokenStream enumerates the sequence of tokens, either from Fields of a Document or from query text.
   * Low-level Classes:
     1. Analyzer.ReuseStrategy: Strategy defining how TokenStreamComponents are reused per call to Analyzer.tokenStream(String, java.io.Reader).
     2. Analyzer.TokenStreamComponents: This class encapsulates the outer components of a token stream.
     3. CharacterUtils: Utility class to write tokenizers or token filters.
     4. CharacterUtils.CharacterBuffer: A simple IO buffer to use with CharacterUtils.fill(CharacterBuffer, Reader).
     5. CharArrayMap: A simple class that stores key Strings as char[]'s in a hash table.
     6. CharArraySet: A simple class that stores Strings as char[]'s in a hash table.
     7. AnalyzerWrapper: Extension to Analyzer suitable for Analyzers which wrap other Analyzers.
     8. CharFilter: Subclasses of CharFilter can be chained to filter a Reader They can be used as Reader with additional offset correction.
     9. StopwordAnalyzerBase: Base class for Analyzers that need to make use of stopword sets.
     10. TokenStreamToAutomaton: Consumes a TokenStream and creates an Automaton where the transition labels are UTF8 bytes (or Unicode code points if unicodeArcs is true) from the TermToBytesRefAttribute.
2. org.apache.lucene.analysis.standard: Fast, general-purpose grammar-based tokenizer StandardTokenizer implements the Word Break rules from the Unicode Text Segmentation algorithm, as specified in Unicode Standard Annex #29.
3. StandardAnalyzer: Filters StandardTokenizer with StandardFilter, LowerCaseFilter and StopFilter, using a list of English stop words.
4. StandardFilter: Normalizes tokens extracted with StandardTokenizer.
5. StandardTokenizer: A grammar-based tokenizer constructed with JFlex.
6. StandardTokenizerImpl: This class implements Word Break rules from the Unicode Text Segmentation algorithm, as specified in Unicode Standard Annex #29.
7. org.apache.lucene.analysis.tokenattributes: General-purpose attributes for text analysis.
   1. BytesTermAttribute: This attribute can be used if you have the raw term bytes to be indexed.’
   2. CharTermAttribute: The term text of a Token.
   3. FlagsAttribute: This attribute can be used to pass different flags down the Tokenizer chain, e.g.
   4. KeywordAttribute: This attribute can be used to mark a token as a keyword.
   5. OffsetAttribute: The start and end character offset of a Token.
   6. PayloadAttribute: The payload of a Token.
   7. PositionIncrementAttribute: Determines the position of this token relative to the previous Token in a TokenStream, used in phrase searching.
   8. PositionLengthAttribute: Determines how many positions this token spans.
   9. TermFrequencyAttribute: Sets the custom term frequency of a term within one document.
   10. TermToBytesRefAttribute: This attribute is requested by TermsHashPerField to index the contents.
   11. TypeAttribute: A Token's lexical type.
   12. BytesTermAttributeImpl: Implementation class for BytesTermAttribute.
   13. CharTermAttributeImpl: Default implementation of CharTermAttribute.
   14. FlagsAttributeImpl: Default implementation of FlagsAttribute.
   15. KeywordAttributeImpl: Default implementation of KeywordAttribute.
   16. OffsetAttributeImpl: Default implementation of OffsetAttribute.
   17. PackedTokenAttributeImpl: Default implementation of the common attributes used by Lucene: CharTermAttribute TypeAttribute PositionIncrementAttribute PositionLengthAttribute OffsetAttribute TermFrequencyAttribute
   18. PayloadAttributeImpl: Default implementation of PayloadAttribute.
   19. PositionIncrementAttributeImpl: Default implementation of PositionIncrementAttribute.
   20. PositionLengthAttributeImpl: Default implementation of PositionLengthAttribute.
   21. TermFrequencyAttributeImpl: Default implementation of TermFrequencyAttribute.
   22. TypeAttributeImpl: Default implementation of TypeAttribute.

## Other:

1. org.apache.lucene: Top-Level Package
2. org.apache.lucene.geo: Geospatial Utility Implementations for Lucene Core

## Summary:

* IR Class:
  + Analyzer: Use for text processing
  + Index: All the index activities and functions such as docFreq….
  + IndexReader: read or process indexes within the index directory
    - leafReader: read individual index within indexReader
* Algorithm Class:
* Data Structure Class:
  + Documents: Data structure of the documents
* Low-Level Class:
  + Codecs: encoding and decoding documents
* Other Class:
  + Geospatial Utility: Geospatial Utility Implementations for Lucene Core