1. Class & Interface Documentation

1.1 `IndexManager` (Interface)

Purpose
`IndexManager` defines the **contract for managing search indexes** for weblogs and weblog entries. It abstracts all indexing-related operations such as initialization, rebuilding indexes, adding/removing entries, and searching.

Key Responsibilities

- Initialize and shut down the indexing system
- Detect index inconsistencies at startup
- Add, remove, and re-index weblog entries
- Rebuild indexes for a single weblog or all weblogs
- Perform search operations and return structured results

Why an interface?
So different indexing implementations (e.g., Lucene-based, database-based) can be swapped without changing higher-level code.

1.2 `SearchResultList`

Purpose
Encapsulates the **results of a search operation**.

Responsibilities

- Store pagination information (`limit`, `offset`)
- Hold search results (`List<WeblogEntryWrapper>`)

- Track categories involved in the search
- Provide getters for controlled access

This class acts as a **data transfer object (DTO)** between the index layer and the application layer.

1.3 `RollerException` (Abstract Class)

Purpose
Base exception class for the system, providing **enhanced exception handling**.

Responsibilities

- Store and expose the root cause of errors
- Provide multiple constructors for flexibility
- Support detailed stack trace printing

Being abstract ensures only meaningful, specific exceptions are thrown.

1.4 `WebloggerException`

Purpose
Represents **general application-level exceptions** in the weblog system.

Responsibilities

- Extend `RollerException`
- Wrap lower-level exceptions with meaningful messages

Used when indexing or weblog-related operations fail.

1.5 `InitializationException`

Purpose
Signals **failures during system or index initialization**.

Responsibilities

- Specialize `WebloggerException`
- Clearly distinguish startup failures from runtime failures

1.6 `URLStrategy` (Interface)

Purpose
Defines a strategy for **constructing URLs** for weblog-related resources.

Responsibilities

- Generate URLs for weblogs, entries, feeds, login, and search
- Support absolute and relative URLs
- Allow preview-based URL strategies

This supports **flexible URL generation** without coupling indexing logic to URL formats.

1.7 `LuceneIndexManager`

Purpose
Concrete implementation of `IndexManager` using **Apache Lucene**.

Responsibilities

- Manage Lucene index readers and writers
- Handle concurrent access using read/write locks

- Execute index operations (add, remove, rebuild)
- Perform search queries and return `SearchResultList`

This class is the **core engine** of the indexing system.

1.8 `IndexOperation` (Abstract Class)

Purpose
Represents a **unit of work** performed on the Lucene index.

Responsibilities

- Provide a template method (`run`)
- Manage writer lifecycle (`beginWriting`, `endWriting`)
- Convert weblog data into Lucene documents
- Delegate actual logic to `doRun()`

Implements the **Template Method pattern**.

1.9 `WriteToIndexOperation` (Abstract Class)

Purpose
Specialization of `IndexOperation` for **write-based operations**.

Responsibilities

- Acquire write locks
- Ensure index consistency during modifications

1.10 `ReadFromIndexOperation` (Abstract Class)

Purpose
Specialization of `IndexOperation` for **read-only operations**.

Responsibilities

- Acquire read locks
- Allow concurrent searches without blocking writers unnecessarily

---

1.11 `AddEntryOperation`

Purpose
Adds a new weblog entry to the index.

Responsibilities

- Convert a `WeblogEntry` into a Lucene document
- Persist it into the index

1.12 `RemoveEntryOperation`

Purpose
Removes an existing weblog entry from the index.

Responsibilities

- Locate the entry document
- Delete it from the Lucene index

1.13 `ReIndexEntryOperation`

Purpose
Re-indexes an existing weblog entry.

Responsibilities

- Remove the old index data
- Add updated index data

Useful when an entry is edited.

1.14 `RebuildWebsiteIndexOperation`

Purpose
Rebuilds the **entire index for a weblog**.

Responsibilities

- Iterate over all entries of a weblog
- Recreate index data from scratch

Used when indexes are corrupted or outdated.

1.15 `SearchOperation`

Purpose
Encapsulates a **search query execution**.

Responsibilities

- Build Lucene queries
- Execute search using `IndexSearcher`

- Store and return `TopFieldDocs`
- Support filters (term, category, locale, weblog)

1.16 `FieldConstants` (Utility Class)

Purpose
Centralizes **Lucene field names** used during indexing and searching.

Responsibilities

- Avoid hardcoded strings
- Ensure consistency across index operations

---

2. Relationship Explanation

2.1 Interface & Inheritance Relationships

- `LuceneIndexManager` **implements** `IndexManager`
  → Provides a Lucene-based implementation.
- `WebloggerException` **extends** `RollerException`
  → Specializes general exception handling.
- `InitializationException` **extends** `WebloggerException`
  → Further specialization for startup errors.
- `IndexOperation` **implements** `Runnable`
  → Allows index operations to run asynchronously or in threads.
- `WriteToIndexOperation` and `ReadFromIndexOperation` **extend** `IndexOperation`
  → Separate read and write concerns for concurrency control.

2.2 Composition & Association

- `IndexManager *-- SearchResultList`
  → Search results are created and owned by the index manager.

- `LuceneIndexManager --> Weblogger`
  → Depends on application context for configuration and data access.

- `IndexOperation --> LuceneIndexManager`
  → Operations execute under the control of the index manager.

- `Add/Remove/ReIndex/Rebuild operations --> Weblog / WeblogEntry`
  → Each operation acts on domain objects.

2.3 Dependency Relationships

- `IndexManager -----> WebloggerException`
  → Index methods may throw application-level exceptions.

- `URLStrategy ..> IndexManager`
  → URL generation supports search results produced by indexing.

- `LuceneIndexManager ..> URLStrategy`
  → Needed to construct URLs in search results.

- `SearchOperation ..> IndexSearcher, TopFieldDocs`
  → Uses Lucene search APIs internally.

- `IndexOperation ..> FieldConstants`
  → Uses consistent field names for indexing/searching.

2.4 Concurrency & Locking Design

- `WriteToIndexOperation ..> LuceneIndexManager : write lock`
  → Prevents concurrent writes and ensures index consistency.

- `ReadFromIndexOperation ..> LuceneIndexManager : read lock`
  → Allows multiple parallel searches while blocking writes.

This design ensures **thread safety and performance**.

---

3. High-Level Design Summary (One-liner)

> The system uses **interfaces for flexibility**, **template methods for consistency**, **operation objects for modularity**, and **read/write locks for concurrency**, making the indexing architecture scalable, maintainable, and robust.
