Analysis & Design Document:

Java-to-UML Diagram Generator

**Author Jags-Programming [**[**jags-programming**](https://github.com/jags-programming)**]**

Contents

[1. Introduction 3](#_Toc193839119)

[1.1 Purpose 3](#_Toc193839120)

[1.2 Scope 3](#_Toc193839121)

[1.3 Users & Stakeholders 3](#_Toc193839122)

[2. Functional Requirements 3](#_Toc193839123)

[2.1 Use Cases & Functionalities 3](#_Toc193839124)

[3. Design Decisions 4](#_Toc193839125)

[3.1 Java Parsing Library Study 4](#_Toc193839126)

[4. System Architecture & Storage Strategy 5](#_Toc193839127)

[4.1 Hybrid Storage Strategy 5](#_Toc193839128)

[4.2 Storage Implementation & Configuration 5](#_Toc193839129)

[4.3 Dynamic Storage Handling 6](#_Toc193839130)

[5. Class Design 7](#_Toc193839131)

[5.1 Core Domain & Software Classes 7](#_Toc193839132)

[5.2 Detailed Design 8](#_Toc193839133)

[5.2.1 UMLDiagramGenerator 11](#_Toc193839134)

[5.2.2 JavaParserService 13](#_Toc193839135)

[5.2.3 ClassDiagramService 14](#_Toc193839136)

[5.2.4 SequenceDiagramService 15](#_Toc193839137)

[5.2.5 FileHandler 16](#_Toc193839138)

[5.2.6 CLIApplication 17](#_Toc193839139)

[5.2.7 GraphicalUI 18](#_Toc193839140)

[5.2.8 CodeEntity 20](#_Toc193839141)

[5.2.9 MethodEntity 21](#_Toc193839142)

[5.2.10 FieldEntity 21](#_Toc193839143)

[5.2.11 RelationshipManager 22](#_Toc193839144)

[5.2.12 ConfigurationManager 22](#_Toc193839145)

[6. Implementation Plan 23](#_Toc193839146)

[6.1 Dependency Tree for the UML Diagram Generator 24](#_Toc193839147)

[Least Dependent (Foundation Classes) – No dependencies 24](#_Toc193839148)

[Low Dependency – Depends on Foundation Classes 24](#_Toc193839149)

[Medium Dependency – Uses Data & Parsing 24](#_Toc193839150)

[High Dependency – Uses Parsed Data & Generates Output 24](#_Toc193839151)

[Most Dependent (Final Application Components) – Uses All Services 24](#_Toc193839152)

[Execution Order Based on Dependencies 25](#_Toc193839153)

[Dependency Tree Graphical Visualization 25](#_Toc193839154)

# 1. Introduction

## 1.1 Purpose

The Java-to-UML Diagram Generator is a tool designed to analyze Java source code and generate UML diagrams, including class diagrams and sequence diagrams. The tool will parse Java code, extract relevant information, and visualize relationships between classes, methods, and interactions.

## 1.2 Scope

* Parse Java source code to extract class structures, methods, relationships, and dependencies.
* Generate UML diagrams in standard formats such as PNG, SVG, and PlantUML.
* Provide both in-memory processing for small projects and persistent storage for large-scale analysis.
* Support integration with CI/CD pipelines for automated UML generation.
* Allow configurable hybrid storage modes to optimize performance and scalability.
* Provide an interactive UI and CLI for ease of use.
* Support cloud storage and integration with online UML tools.

## 1.3 Users & Stakeholders

* **Software Developers**: Understand and document system architecture.
* **Architects**: Analyze and refine system design.
* **QA Teams**: Validate system behavior through sequence diagrams.
* **Project Managers**: Gain insights into project structure and dependencies.

# 2. Functional Requirements

## 2.1 Use Cases & Functionalities

| **Use Case** | **Description** |
| --- | --- |
| **Parse Java Code** | Read and analyze Java source files to extract classes, methods, and relationships. |
| **Generate Class Diagrams** | Convert parsed data into UML class diagrams. |
| **Generate Sequence Diagrams** | Identify method calls and interactions to create sequence diagrams. |
| **Support Multiple Outputs** | Export diagrams as PNG, SVG, and PlantUML code. |
| **Incremental Updates** | Track changes in code and update UML diagrams accordingly. |
| **Persistence & Storage** | Store parsed data in memory or a database for large projects. |
| **Integration with CI/CD** | Generate UML diagrams automatically in a CI/CD pipeline. |
| **Hybrid Storage Configuration** | Allow users to switch between in-memory and persistent storage dynamically. |
| **Interactive UI** | Provide a graphical user interface for easy diagram visualization. |
| **Cloud Integration** | Store and retrieve UML diagrams from cloud-based services. |
| **Code Annotation Support** | Recognize annotations and include them in UML diagrams. |

# 3. Design Decisions

## 3.1 Java Parsing Library Study

Three primary Java parsing libraries were analyzed:

| **Library** | **Pros** | **Cons** |
| --- | --- | --- |
| **JavaParser** | Easy to use, actively maintained, lightweight | Limited support for complex AST manipulations |
| **Eclipse JDT** | Powerful, supports full AST manipulation | Requires Eclipse dependencies, more complex |
| **Spoon** | Rich transformation capabilities, works well for refactoring | Heavier learning curve |

👉 **Decision**: JavaParser is chosen for its simplicity and efficiency. Eclipse JDT may be considered for future enhancements.

# 4. System Architecture & Storage Strategy

## 4.1 Hybrid Storage Strategy

A hybrid approach will be used:

* **Small & Medium Projects** → Process everything in-memory for speed and simplicity.
* **Large Projects** → Store parsed data persistently to avoid repeated parsing overhead.
* **Cloud Storage** → Optionally store UML diagrams in cloud repositories.
* **Configurable Storage Mode** → Users can specify their preference in a configuration file or CLI flag.

## 4.2 Storage Implementation & Configuration

| **Option** | **Pros** | **Cons** | **Best Use Case** |
| --- | --- | --- | --- |
| **In-Memory** | Fast, no setup required | Cannot persist data across sessions | Small projects & quick analysis |
| **JSON/XML Files** | Human-readable, easy to export/import | Slower querying | Small projects requiring persistence |
| **SQLite/PostgreSQL** | Good for structured data, supports indexing and querying | More complex setup | Medium to large projects needing fast lookups |
| **NoSQL (MongoDB, Redis)** | Fast lookups, flexible schema | Not ideal for complex queries | High-speed access required |
| **Cloud Storage (AWS S3, Google Drive)** | Remote access, backup | Requires API setup, latency concerns | Distributed teams & backup needs |

👉 **Implementation:**

* Default to **in-memory processing**.
* Store **parsed data in JSON** for small projects.
* Use **SQLite/PostgreSQL** for large projects requiring persistence.
* Allow users to specify their preferred storage method via a configuration file:
* {
* "storage\_mode": "memory", // Options: "memory", "json", "database", "cloud"
* "database": {
* "type": "sqlite",
* "connection\_string": "uml\_data.db"
* },
* "cloud": {
* "provider": "AWS S3",
* "bucket\_name": "uml-diagrams"
* }
* }

## 4.3 Dynamic Storage Handling

To support hybrid storage, a **StorageService** will be implemented using the **Strategy Pattern**:

interface StorageStrategy {

void saveParsedData(ParsedData data);

ParsedData loadParsedData();

}

class InMemoryStorage implements StorageStrategy {

private Map<String, ParsedData> cache = new HashMap<>();

public void saveParsedData(ParsedData data) { cache.put(data.getId(), data); }

public ParsedData loadParsedData() { return cache.get(data.getId()); }

}

class CloudStorage implements StorageStrategy {

public void saveParsedData(ParsedData data) { /\* Upload to cloud \*/ }

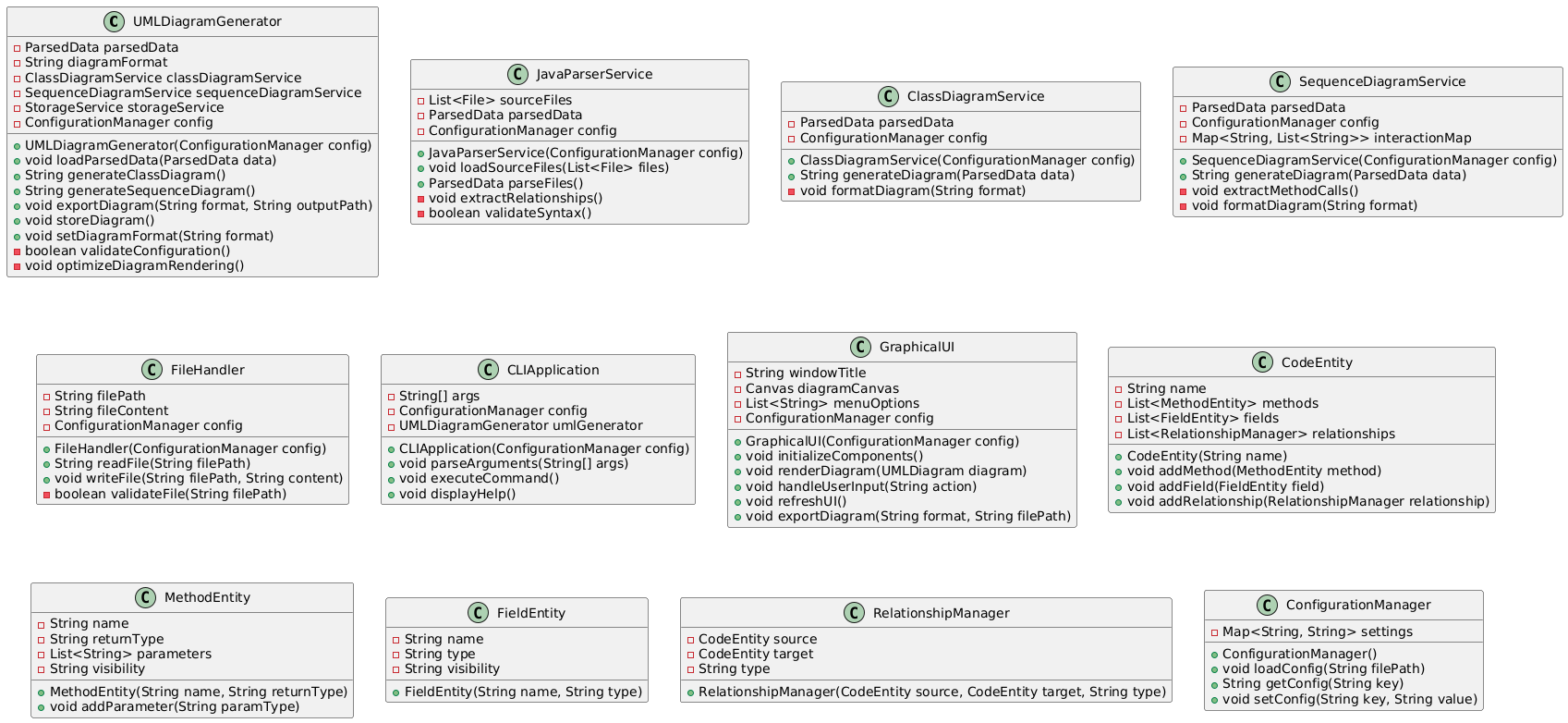
public ParsedData loadParsedData() { /\* Retrieve from cloud \*/ }

}

The system will dynamically select a storage strategy based on user configuration.

# 5. Class Design

## 5.1 Core Domain & Software Classes



| **Class** | **Responsibility** |
| --- | --- |
| **JavaParserService** | Parses Java source code and extracts metadata. |
| **UMLDiagramGenerator** | Converts extracted data into UML diagrams. |
| **ClassDiagramService** | Handles generation of class diagrams. |
| **SequenceDiagramService** | Handles generation of sequence diagrams. |
| **StorageService** | Manages in-memory, cloud, and database storage dynamically. |
| **FileHandler** | Reads and writes Java files. |
| **CLIApplication** | Provides command-line interface for running the tool. |
| **GraphicalUI** | Provides an interactive UI for users. |
| **CodeEntity** | Represents a Java class or interface. |
| **MethodEntity** | Represents a method within a class. |
| **FieldEntity** | Represents a field within a class. |
| **RelationshipManager** | Analyzes and manages relationships between classes. |
| **ConfigurationManager** | Loads and manages user preferences and storage configurations. |

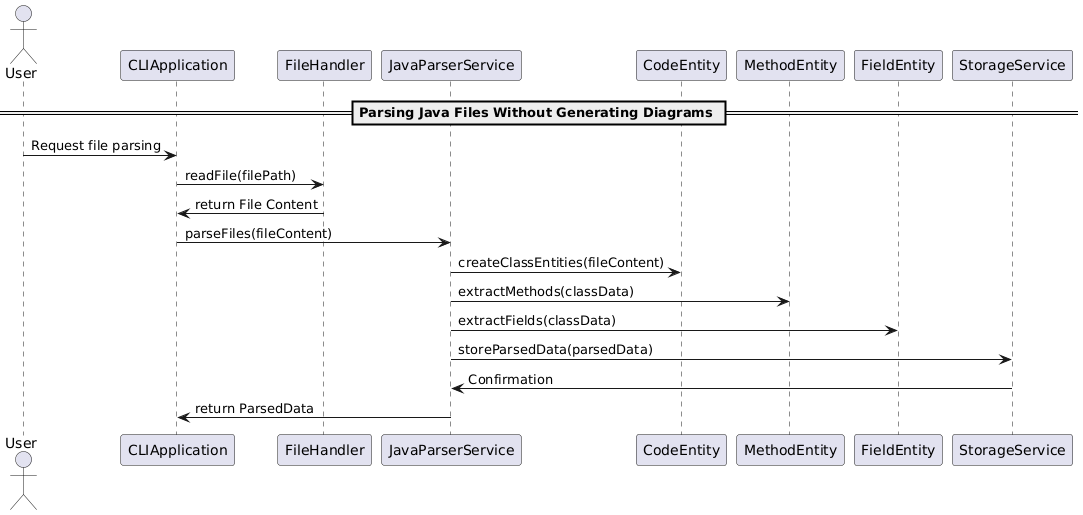
## 5.2 Detailed Design

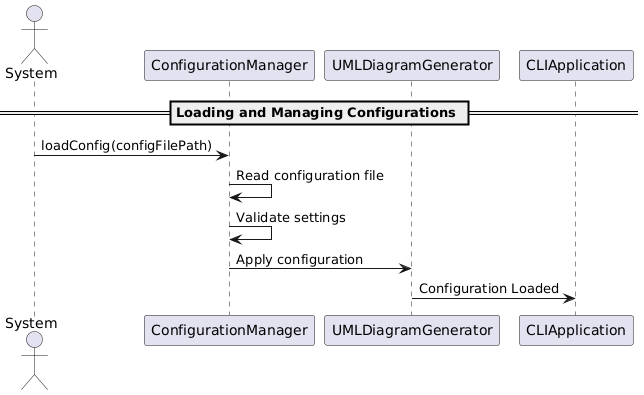
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.





A diagram of exporting uml diagrams

AI-generated content may be incorrect.

A diagram of a program

AI-generated content may be incorrect.

### 5.2.1 UMLDiagramGenerator

#### 5.2.1.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **parsedData** | **ParsedData** | **Private** | Holds extracted Java code metadata (classes, methods, relationships). |
| **diagramFormat** | **String** | **Private** | Defines the output format (e.g., PNG, SVG, PlantUML). |
| **classDiagramService** | **ClassDiagramService** | **Private** | Service for generating UML class diagrams. |
| **sequenceDiagramService** | **SequenceDiagramService** | **Private** | Service for generating UML sequence diagrams. |
| **storageService** | **StorageService** | **Private** | Handles diagram storage (local, database, cloud). |
| **config** | **ConfigurationManager** | **Private** | Manages user-defined settings and configurations. |

#### 5.2.1.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **UMLDiagramGenerator(ConfigurationManager config)** | **Public** | **Constructor** | Initializes the generator with configuration settings. |
| **loadParsedData(ParsedData data)** | **Public** | **void** | Loads extracted Java code metadata. |
| **generateClassDiagram()** | **Public** | **String** | Calls ClassDiagramService to generate a class diagram and returns its representation. |
| **generateSequenceDiagram()** | **Public** | **String** | Calls SequenceDiagramService to generate a sequence diagram and returns its representation. |
| **exportDiagram(String format, String outputPath)** | **Public** | **void** | Exports UML diagrams in a specified format to a file. |
| **storeDiagram()** | **Public** | **void** | Saves the generated UML diagram using StorageService. |
| **setDiagramFormat(String format)** | **Public** | **void** | Allows users to change the UML output format dynamically. |
| **validateConfiguration()** | **Private** | **boolean** | Ensures required settings are correctly configured before generation. |
| **optimizeDiagramRendering()** | **Private** | **void** | Enhances rendering efficiency for large diagrams. |

#### 5.2.1.3 Collaborators

* **ClassDiagramService (generates class diagrams)**
* **SequenceDiagramService (generates sequence diagrams)**
* **StorageService (handles storage of generated diagrams)**
* **ConfigurationManager (retrieves user settings)**

#### 5.2.1.4 Internal Collaboration Between Class Methods

* **loadParsedData() loads extracted Java code data and prepares it for diagram generation.**
* **generateClassDiagram() and generateSequenceDiagram() call respective services based on user selection.**
* **exportDiagram() ensures the correct format is applied before storing the output.**
* **validateConfiguration() runs before processing to check if the required settings are met.**
* **optimizeDiagramRendering() is invoked when handling large projects to enhance performance.**

### 5.2.2 JavaParserService

#### 5.2.2.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **sourceFiles** | **List<File>** | **Private** | Stores Java source files to be parsed. |
| **parsedData** | **ParsedData** | **Private** | Holds extracted metadata after parsing. |
| **config** | **ConfigurationManager** | **Private** | Handles parsing-related settings. |

#### 5.2.2.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **JavaParserService(ConfigurationManager config)** | **Public** | **Constructor** | Initializes the parser with configuration settings. |
| **loadSourceFiles(List<File> files)** | **Public** | **void** | Loads Java source files for parsing. |
| **parseFiles()** | **Public** | **ParsedData** | Parses loaded Java files and extracts class structures. |
| **extractRelationships()** | **Private** | **void** | Identifies inheritance, association, and dependencies. |
| **validateSyntax()** | **Private** | **boolean** | Checks for valid Java syntax before parsing. |

#### 5.2.2.3 Collaborators

* **ConfigurationManager (retrieves parsing-related settings)**
* **UMLDiagramGenerator (passes extracted metadata for UML generation)**
* **StorageService (stores parsed data if needed)**

#### 5.2.2.4 Internal Collaboration Between Class Methods

* **loadSourceFiles() loads files into memory before parsing.**
* **parseFiles() processes each file and extracts classes, methods, and fields.**
* **extractRelationships() identifies dependencies between classes and stores them in parsedData.**
* **validateSyntax() ensures the Java files are correctly formatted before parsing.**

### 5.2.3 ClassDiagramService

#### 5.2.3.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **parsedData** | **ParsedData** | **Private** | Holds parsed metadata from Java source files. |
| **config** | **ConfigurationManager** | **Private** | Stores user preferences for class diagrams. |

#### 5.2.3.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **ClassDiagramService(ConfigurationManager config)** | **Public** | **Constructor** | Initializes the service with user preferences. |
| **generateDiagram(ParsedData data)** | **Public** | **String** | Generates a class diagram based on parsed metadata. |
| **formatDiagram(String format)** | **Private** | **void** | Converts the diagram to the desired output format. |

#### 5.2.3.3 Collaborators

* **UMLDiagramGenerator (provides parsed data)**
* **StorageService (stores generated class diagrams)**
* **ConfigurationManager (retrieves user settings)**

#### 5.2.3.4 Internal Collaboration Between Class Methods

* **generateDiagram() takes parsed data and converts it into a structured UML format.**
* **formatDiagram() ensures the output adheres to the specified format before storage.**

### 5.2.4 SequenceDiagramService

#### 5.2.4.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **parsedData** | **ParsedData** | **Private** | Stores extracted Java metadata including method interactions. |
| **config** | **ConfigurationManager** | **Private** | Stores user preferences for sequence diagrams. |
| **interactionMap** | **Map<String, List<String>>** | **Private** | Tracks method calls between different classes. |

#### 5.2.4.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **SequenceDiagramService(ConfigurationManager config)** | **Public** | **Constructor** | Initializes the service with user preferences. |
| **generateDiagram(ParsedData data)** | **Public** | **String** | Generates a sequence diagram based on method interactions. |
| **extractMethodCalls()** | **Private** | **void** | Extracts method calls and builds interaction relationships. |
| **formatDiagram(String format)** | **Private** | **void** | Converts the diagram to the desired output format. |

#### 5.2.4.3 Collaborators

* **UMLDiagramGenerator (provides parsed method interaction data)**
* **StorageService (stores generated sequence diagrams)**
* **ConfigurationManager (retrieves user settings)**
* **JavaParserService (extracts method calls from Java code)**

#### 5.2.4.4 Internal Collaboration Between Class Methods

* **generateDiagram() takes parsed data and converts it into a structured UML format.**
* **extractMethodCalls() scans method definitions and interactions, building an internal representation of calls.**
* **formatDiagram() ensures the output adheres to the specified format before storage.**

### 5.2.5 FileHandler

#### 5.2.5.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **filePath** | **String** | **Private** | Stores the file path of the Java source or output file. |
| **fileContent** | **String** | **Private** | Stores the content of the file read into memory. |
| **config** | **ConfigurationManager** | **Private** | Stores user preferences related to file handling. |

#### 5.2.5.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **FileHandler(ConfigurationManager config)** | **Public** | **Constructor** | Initializes the file handler with configuration settings. |
| **readFile(String filePath)** | **Public** | **String** | Reads a file from the specified path and returns its content. |
| **writeFile(String filePath, String content)** | **Public** | **void** | Writes the given content to the specified file path. |
| **validateFile(String filePath)** | **Private** | **boolean** | Checks if the file exists and is accessible. |

#### 5.2.5.3 Collaborators

* **JavaParserService (reads Java source files for parsing)**
* **UMLDiagramGenerator (writes UML output files to disk)**
* **StorageService (manages saving and retrieving files from storage)**
* **ConfigurationManager (retrieves file handling settings)**

#### 5.2.5.4 Internal Collaboration Between Class Methods

* **readFile() checks file validity using validateFile() before reading the content.**
* **writeFile() ensures proper file path formatting before writing data.**
* **validateFile() is called internally by both read and write operations to prevent errors.**

### 5.2.6 CLIApplication

#### 5.2.6.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **args** | **String[]** | **Private** | **Stores command-line arguments passed by the user.** |
| **config** | **ConfigurationManager** | **Private** | **Stores user preferences for CLI operations.** |
| **umlGenerator** | **UMLDiagramGenerator** | **Private** | **Handles the generation of UML diagrams.** |

#### 5.2.6.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **CLIApplication(ConfigurationManager config)** | **Public** | **Constructor** | **Initializes the CLI with configuration settings.** |
| **parseArguments(String[] args)** | **Public** | **void** | **Parses command-line arguments for user commands.** |
| **executeCommand()** | **Public** | **void** | **Executes the requested command based on parsed arguments.** |
| **displayHelp()** | **Public** | **void** | **Displays available CLI commands and usage information.** |

#### 5.2.6.3 Collaborators

* **UMLDiagramGenerator (triggers UML generation)**
* **FileHandler (handles file operations for input and output)**
* **StorageService (manages saving and loading UML diagrams)**
* **ConfigurationManager (retrieves user CLI settings)**

#### 5.2.6.4 Internal Collaboration Between Class Methods

* **parseArguments() reads user input and sets command parameters.**
* **executeCommand() processes parsed commands and calls umlGenerator for diagram generation.**
* **displayHelp() provides user-friendly CLI documentation.**

### 5.2.7 GraphicalUI

#### 5.2.7.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **windowTitle** | **String** | **Private** | **The title of the UI window.** |
| **diagramCanvas** | **Canvas** | **Private** | **Area where UML diagrams are rendered.** |
| **menuOptions** | **List<String>** | **Private** | **Stores available UI menu options.** |
| **config** | **ConfigurationManager** | **Private** | **Stores user preferences related to UI.** |

#### 5.2.7.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **GraphicalUI(ConfigurationManager config)** | **Public** | **Constructor** | **Initializes the graphical UI with user settings.** |
| **initializeComponents()** | **Private** | **void** | **Sets up UI components like menu, canvas, and panels.** |
| **renderDiagram(UMLDiagram diagram)** | **Public** | **void** | **Displays a UML diagram on the canvas.** |
| **handleUserInput(String action)** | **Public** | **void** | **Processes user interactions like zoom, save, or edit.** |
| **refreshUI()** | **Public** | **void** | **Refreshes the UI after updates.** |
| **exportDiagram(String format, String filePath)** | **Public** | **void** | **Exports the displayed diagram to a file.** |

#### 5.2.7.3 Collaborators

* **UMLDiagramGenerator (provides UML diagrams for rendering)**
* **StorageService (handles exporting diagrams)**
* **ConfigurationManager (retrieves UI settings)**

#### 5.2.7.4 Internal Collaboration Between Class Methods

* **initializeComponents() sets up the UI before rendering any diagrams.**
* **renderDiagram() updates the canvas whenever a new UML diagram is available.**
* **handleUserInput() processes user actions such as zooming, exporting, or navigating between diagrams.**
* **exportDiagram() saves the currently displayed diagram based on user preferences.**
* **refreshUI() ensures that changes (e.g., new diagram loaded) are reflected visually.**

### 5.2.8 CodeEntity

#### 5.2.8.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **name** | **String** | **Private** | **Name of the Java class or interface.** |
| **methods** | **List<MethodEntity>** | **Private** | **List of methods in the class.** |
| **fields** | **List<FieldEntity>** | **Private** | **List of fields in the class.** |
| **relationships** | **List<RelationshipManager>** | **Private** | **Stores relationships with other classes.** |

#### 5.2.8.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **CodeEntity(String name)** | **Public** | **Constructor** | **Initializes a CodeEntity with a name.** |
| **addMethod(MethodEntity method)** | **Public** | **void** | **Adds a method to the class.** |
| **addField(FieldEntity field)** | **Public** | **void** | **Adds a field to the class.** |
| **addRelationship(RelationshipManager relationship)** | **Public** | **void** | **Adds a relationship with another class.** |

#### 5.2.8.3 Collaborators

* **MethodEntity (stores method details)**
* **FieldEntity (stores field details)**
* **RelationshipManager (manages class dependencies)**

#### 5.2.8.4 Internal Collaboration Between Class Methods

* **addMethod(), addField(), and addRelationship() allow structured additions to a class.**

### 5.2.9 MethodEntity

#### 5.2.9.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **name** | **String** | **Private** | **Method name.** |
| **returnType** | **String** | **Private** | **Return type of the method.** |
| **parameters** | **List<String>** | **Private** | **List of parameter types.** |
| **visibility** | **String** | **Private** | **Access modifier (public, private, etc.).** |

#### 5.2.9.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **MethodEntity(String name, String returnType)** | **Public** | **Constructor** | **Initializes a method entity.** |
| **addParameter(String paramType)** | **Public** | **void** | **Adds a parameter to the method.** |

#### 5.2.9.3 Collaborators

* **CodeEntity (associated with a class)**

#### 5.2.9.4 Internal Collaboration Between Class Methods

* **addParameter() dynamically updates method parameters.**

### 5.2.10 FieldEntity

#### 5.2.10.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **name** | **String** | **Private** | **Field name.** |
| **type** | **String** | **Private** | **Data type of the field.** |
| **visibility** | **String** | **Private** | **Access modifier.** |

#### 5.2.10.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **FieldEntity(String name, String type)** | **Public** | **Constructor** | **Initializes a field entity.** |

#### 5.2.10.3 Collaborators

* **CodeEntity (stores field details)**

#### 5.2.10.4 Internal Collaboration Between Class Methods

* **Fields are assigned and retrieved within CodeEntity.**

### 5.2.11 RelationshipManager

#### 5.2.11.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **source** | **CodeEntity** | **Private** | **The class that owns this relationship.** |
| **target** | **CodeEntity** | **Private** | **The class being referenced.** |
| **type** | **String** | **Private** | **Type of relationship (inheritance, association, etc.).** |

#### 5.2.11.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **RelationshipManager(CodeEntity source, CodeEntity target, String type)** | **Public** | **Constructor** | **Initializes a relationship.** |

#### 5.2.11.3 Collaborators

* **CodeEntity (defines class relationships)**

#### 5.2.11.4 Internal Collaboration Between Class Methods

* **Relationship objects are managed and retrieved within CodeEntity.**

### 5.2.12 ConfigurationManager

#### 5.2.12.1 Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Scope** | **Description** |
| **settings** | **Map<String, String>** | **Private** | **Stores configuration settings as key-value pairs.** |

#### 5.2.12.2 Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Scope** | **Return Type** | **Description** |
| **ConfigurationManager()** | **Public** | **Constructor** | **Initializes an empty configuration.** |
| **loadConfig(String filePath)** | **Public** | **void** | **Loads settings from a file.** |
| **getConfig(String key)** | **Public** | **String** | **Retrieves a setting value.** |
| **setConfig(String key, String value)** | **Public** | **void** | **Updates a setting value.** |

#### 5.2.12.3 Collaborators

* **All services (manages configuration settings)**

#### 5.2.12.4 Internal Collaboration Between Class Methods

* **loadConfig() loads settings on initialization.**
* **getConfig() and setConfig() manage dynamic updates.**

# 6. Implementation Plan

* **Phase 1: Implement in-memory processing for initial prototypes.**
* **Phase 2: Introduce a persistence layer (SQLite/PostgreSQL).**
* **Phase 3: Optimize with incremental updates and caching.**
* **Phase 4: Implement hybrid mode switching via configuration.**
* **Phase 5: Add cloud storage support.**
* **Phase 6: Develop the graphical UI for better user interaction.**

## 6.1 Dependency Tree for the UML Diagram Generator

**Below is the dependency order from least dependent (independent classes) to most dependent (highly integrated classes).**

### Least Dependent (Foundation Classes) – No dependencies

**These classes define core data structures and don't rely on other components.**

* **ConfigurationManager → Loads and manages system settings.**
* **FieldEntity → Represents a class field.**
* **MethodEntity → Represents a method.**
* **CodeEntity → Represents a Java class or interface.**

### Low Dependency – Depends on Foundation Classes

**These classes interact with basic data structures.**

* **RelationshipManager → Manages dependencies between CodeEntity objects.**
* **FileHandler → Reads/writes Java files (but doesn’t interact with diagrams).**

### Medium Dependency – Uses Data & Parsing

**These classes rely on file handling, parsing, and configuration.**

* **JavaParserService → Parses Java source code, uses FileHandler, CodeEntity, MethodEntity, and FieldEntity.**
* **StorageService → Stores UML diagrams, depends on ConfigurationManager.**

### High Dependency – Uses Parsed Data & Generates Output

**These classes depend on parsed Java data and configuration settings.**

* **ClassDiagramService → Uses JavaParserService and RelationshipManager.**
* **SequenceDiagramService → Uses JavaParserService and method relationships.**

### Most Dependent (Final Application Components) – Uses All Services

**These classes integrate everything and provide the final tool functionality.**

* **UMLDiagramGenerator → Uses ClassDiagramService, SequenceDiagramService, and StorageService.**
* **CLIApplication → Uses UMLDiagramGenerator and ConfigurationManager for command-line execution.**
* **GraphicalUI → Uses UMLDiagramGenerator, StorageService, and ConfigurationManager to provide a user interface.**

## Execution Order Based on Dependencies

**To build this project in phases, follow this order:**

* **Start with ConfigurationManager, FieldEntity, MethodEntity, CodeEntity (core data structures).**
* **Build RelationshipManager and FileHandler (manages dependencies and file I/O).**
* **Develop JavaParserService and StorageService (parsing and storing data).**
* **Implement ClassDiagramService and SequenceDiagramService (generate UML).**
* **Finalize UMLDiagramGenerator, CLIApplication, and GraphicalUI (main application logic).**

## Dependency Tree Graphical Visualization

**A diagram of a network

AI-generated content may be incorrect.**

A diagram of a computer

AI-generated content may be incorrect.