Organization Analyzer

A Java application that analyzes organizational structure to identify salary compliance issues and overly long reporting lines.

Problem Statement

BIG COMPANY needs to ensure:

1. Every manager earns **at least 20% more** than the average salary of their direct subordinates
2. Every manager earns **no more than 50% more** than the average salary of their direct subordinates
3. No employee has **more than 4 managers** between them and the CEO

Business Rules Explained

Rule 1: Minimum Manager Salary (20% above team average)

Purpose: Prevents underpaid managers

Formula:

```
Manager Salary ≥ 1.20 × Average(Subordinate Salaries)
```

Example:

```
Subordinates earn: 40,000; 50,000; 60,000
Average = (40,000 + 50,000 + 60,000) / 3 = 50,000

Manager must earn at least: 1.20 × 50,000 = 60,000

✗ Manager earns 55,000 → UNDERPAID (needs 5,000 more)
✓ Manager earns 65,000 → OK
```

Rule 2: Maximum Manager Salary (50% above team average)

Purpose: Prevents excessive pay gaps between managers and their teams

Formula:

```
Manager Salary ≤ 1.50 × Average(Subordinate Salaries)
```

Example:

```
Same subordinates: average = 50,000
Manager must earn at most: 1.50 × 50,000 = 75,000

✗ Manager earns 80,000 → OVERPAID (by 5,000)
✓ Manager earns 72,000 → OK
```

Combined Salary Band

Rules 1 & 2 together create an acceptable salary range for managers:

```
Using average subordinate salary of 50,000:

Minimum salary: 60,000  (20% above average)
Maximum salary: 75,000  (50% above average)

Valid range: 60,000 – 75,000
```

Rule 3: Maximum Reporting Line Depth (≤ 4 managers to CEO)

Purpose: Keeps organizational hierarchy flat and efficient

Why this matters:

- Communication becomes slow in deep hierarchies
- Decision making becomes inefficient
- Company structure becomes too vertical

Allowed Structure (4 managers):

```
CEO
 └─ Manager 1
     └─ Manager 2
         └─ Manager 3
             └─ Manager 4
                 └─ Employee  ✓ (4 managers above – OK)
```

Not Allowed (5 managers):

```
CEO
 └─ Manager 1
     └─ Manager 2
         └─ Manager 3
             └─ Manager 4
                 └─ Manager 5
                     └─ Employee  ✗ (5 managers above – TOO DEEP)
```

Rules Summary

Rule	Description	Purpose
1	Manager earns ≥ 20% more than team average	Prevents underpaid managers
2	Manager earns ≤ 50% more than team average	Prevents excessive pay gaps
3	Depth ≤ 4 managers between employee & CEO	Keeps hierarchy flat and efficient

Approach and Design

Class Diagram

The PlantUML source is available at docs/class-diagram.puml.

To generate the diagram:

```
# Using PlantUML CLI
java -jar plantuml.jar docs/class-diagram.puml

# Or use online: https://www.plantuml.com/plantuml
```

Architecture

The application follows a clean, layered architecture with clear separation of concerns:

```
com.bigcompany.analyzer
├── model/           # Domain entities (immutable)
│   └── Employee     # Core employee representation
├── parser/          # Data access layer
│   ├── EmployeeCsvParser      # CSV file parsing with validation
│   └── CsvParseException      # Custom exception for parse errors
├── service/         # Business logic layer
│   ├── OrganizationAnalyzer   # Core analysis engine
│   ├── SalaryAnalysisResult   # Salary analysis result (record)
│   ├── ReportingLineResult    # Reporting line result (record)
│   └── AnalysisReportPrinter  # Console report formatting
└── Application      # Entry point
```

Design Decisions

Decision	Rationale
Immutable Employee Model	Prevents invalid state, thread-safe, validated in constructor
Java Records for Results	Immutable by default, reduces boilerplate, clear semantics
Separation of Concerns	Parser handles I/O, Analyzer handles logic, Printer handles output
Custom Exceptions	Detailed error messages with line numbers for debugging
Stream API	Clean, functional approach for collection processing
No External Libraries	Uses only Java SE as per requirements

Key Algorithms

Salary Analysis

```
For each manager:
  1. Find all direct subordinates (employees with managerId = manager.id)
  2. Calculate average salary of subordinates
  3. Calculate expected range:
     - minExpected = average × 1.20
     - maxExpected = average × 1.50
  4. Compare actual salary against range
```

Reporting Line Analysis

```
For each employee:
  1. Start from employee, follow managerId chain to CEO
  2. Count number of managers traversed
  3. Detect circular references (visited set)
  4. Flag if count > 4
```

Assumptions

Assumption	Explanation
CSV has header row	First row is always `Id,firstName,lastName,salary,managerId`
Single CEO	Exactly one employee has no manager
Valid manager references	All managerId values reference existing employees
Positive salaries	Salaries are non-negative numeric values
"More than 4 managers"	Reporting line length > 4 means too long

Prerequisites

- Java 17 or higher
- Maven 3.6 or higher
- Docker (optional, for containerized deployment)

Building the Project

```
cd organization-analyzer
mvn clean package
```

Running the Application

```
# Using Maven
mvn exec:java -Dexec.mainClass="com.bigcompany.analyzer.Application" \
    -Dexec.args="src/main/resources/employees.csv"

# Using the JAR file
java -jar target/organization-analyzer-1.0.0.jar <path-to-csv-file>

# Example with included sample data
java -jar target/organization-analyzer-1.0.0.jar src/main/resources/employees.csv
```

Sample Output

```
Successfully loaded 80 employees.

======================================================================
              ORGANIZATION STRUCTURE ANALYSIS REPORT
======================================================================

----------------------------------------------------------------------
MANAGERS EARNING LESS THAN REQUIRED (below 20% above subordinate avg)
----------------------------------------------------------------------
  Manager Name              | Current Salary  | Min Required    | Underpaid By
  ------------------------------------------------------------------
  John Smith                |      250,000.00 |      276,000.00 |     26,000.00

----------------------------------------------------------------------
MANAGERS EARNING MORE THAN ALLOWED (above 50% above subordinate avg)
----------------------------------------------------------------------
  No issues found.

----------------------------------------------------------------------
EMPLOYEES WITH REPORTING LINE TOO LONG (more than 4 managers to CEO)
----------------------------------------------------------------------
  Employee Name                   | Line Length     | Excess
  ------------------------------------------------------------
  Richard Kim                     |               5 |          1

======================================================================
                         END OF REPORT
======================================================================
```

Docker

Building the Docker Image

```
docker build -t organization-analyzer:1.0.0 .
```

Running with Docker

```
# Run with included sample data
docker run organization-analyzer:1.0.0

# Run with custom CSV file (mount your data directory)
docker run -v /path/to/your/data:/app/data organization-analyzer:1.0.0 /app/data/your-f

# Example with absolute path
docker run -v $(pwd)/mydata:/app/data organization-analyzer:1.0.0 /app/data/employees.c
```

Docker Image Details

Property	Value
Base Image	`eclipse-temurin:17-jre-alpine`
Image Size	~200MB
Build	Multi-stage (Maven build → JRE runtime)
Default Data	`/app/data/employees.csv` (80 employees)

Running Tests

```
# Run all tests
mvn test

# Run only data-driven tests
mvn test -Dtest=DataDrivenOrganizationAnalyzerTest

# Run with verbose output
mvn test -Dsurefire.reportFormat=brief
```

Test Architecture

The project uses a **data-driven testing approach** for comprehensive coverage:

Test Structure

```
src/test/
├── java/
│   └── com/bigcompany/analyzer/
│       ├── DataDrivenOrganizationAnalyzerTest.java  # Parameterized test runner
│       ├── model/
│       │   └── EmployeeTest.java                    # Unit tests for Employee
│       ├── parser/
│       │   └── EmployeeCsvParserTest.java           # Unit tests for parser
│       └── service/
│           ├── OrganizationAnalyzerTest.java        # Unit tests for analyzer
│           └── TestResultFormatter.java             # Test output formatter
└── resources/
    └── testcases/                                   # Data-driven test cases
        ├── 01_underpaid_manager/
        │   ├── input.csv                            # Test input
        │   └── expected.txt                         # Expected output
        ├── 02_overpaid_manager/
        │   ├── input.csv
        │   └── expected.txt
        └── ... (24 test cases total)
```

Test Case Categories

Category	Test Cases	Description
Salary - Underpaid	01, 09	Managers earning < 20% above subordinate avg
Salary - Overpaid	02, 10	Managers earning > 50% above subordinate avg
Salary - Compliant	03, 07, 08	Managers within valid range
Reporting Lines	04, 05, 14	Long chains, boundary cases
Structure	06, 12, 13	CEO only, single subordinate, wide org
Edge Cases	15, 16	Decimal salaries, zero salaries
Complex	11, 17	Multiple issues combined
Errors	error_01-07	Circular refs, missing data, parse errors

Adding New Test Cases

1. Create a new directory under `src/test/resources/testcases/`
2. Add `input.csv` with employee data
3. Add `expected.txt` with expected output
4. Run tests - new case is automatically discovered

Expected Output Format

For success cases:

```
UNDERPAID_MANAGERS:
Name|ActualSalary|MinExpected|Deviation
OVERPAID_MANAGERS:
Name|ActualSalary|MaxExpected|Deviation
LONG_REPORTING_LINES:
Name|LineLength|Excess
```

For error cases:

```
ERROR:Circular reference detected
PARSE_ERROR:Invalid salary
```

CSV File Format

```
Id,firstName,lastName,salary,managerId
123,Joe,Doe,60000,
124,Martin,Chekov,45000,123
125,Bob,Ronstad,47000,123
```

Column	Description	Required
Id	Unique employee identifier	Yes
firstName	Employee's first name	Yes
lastName	Employee's last name	Yes
salary	Employee's salary (numeric)	Yes

Column	Description	Required
managerId	ID of employee's manager	No (empty for CEO)

Project Structure

```
organization-analyzer/
├── pom.xml                    # Maven build configuration
├── Dockerfile                 # Docker build configuration
├── .dockerignore              # Docker build exclusions
├── README.md                  # This file
└── src/
    ├── main/
    │   ├── java/
    │   │   └── com/bigcompany/analyzer/
    │   │       ├── Application.java
    │   │       ├── model/
    │   │       │   └── Employee.java
    │   │       ├── parser/
    │   │       │   ├── CsvParseException.java
    │   │       │   └── EmployeeCsvParser.java
    │   │       └── service/
    │   │           ├── AnalysisReportPrinter.java
    │   │           ├── OrganizationAnalyzer.java
    │   │           ├── ReportingLineResult.java
    │   │           └── SalaryAnalysisResult.java
    │   └── resources/
    │       └── employees.csv   # Sample data (80 employees)
    └── test/
        ├── java/               # Test classes
        └── resources/
            └── testcases/      # 24 data-driven test cases
```

Key Features

Feature	Description
Modular Design	Easy to extend with new analysis rules or output formats
Data-Driven Tests	52 tests covering all edge cases via input/output files

Feature	Description
Comprehensive Validation	Detailed error messages with line numbers
Clean Code	JavaDoc documentation, follows Java conventions
No External Dependencies	Uses only Java SE (except JUnit for testing)
Circular Reference Detection	Prevents infinite loops in org structure

License