**Smart Resume Analyzer & Ranker - Project Blueprint**

**Project Overview**

**Project Type: Core Java (Intermediate Level)  
Goal: Develop a resume ranking system using Java OOP concepts, multithreading, collections, file handling, and design patterns.**

**Features**

* **Read multiple resumes (PDF/DOCX format)**
* **Extract candidate details (name, contact, skills, experience, education)**
* **Compare extracted skills with job requirements**
* **Rank resumes based on skills and experience**
* **Display results in CLI or GUI format**

**Concepts Covered**

* **OOP Principles: Encapsulation, Inheritance, Polymorphism, Abstraction**
* **Interfaces & Abstract Classes: Define common behavior for resume parsing**
* **Java Collections: Lists, Sets, Maps, Comparator for sorting**
* **File Handling: Java I/O for reading resumes**
* **Multithreading: Process multiple resumes in parallel**
* **Design Patterns: Factory Pattern (for different resume formats), Singleton Pattern (for configuration management)**
* **Exception Handling: Handle invalid/missing files**
* **Regex & String Manipulation: Extract structured data from text**

**System Architecture**

**Project Structure**

**📂 ResumeAnalyzer**

**├── 📂 src**

**│ ├── 📂 models # Data Models (Candidate, Resume)**

**│ │ ├── Candidate.java**

**│ │ ├── Resume.java**

**│ │ ├── AbstractResumeParser.java (Abstract Class)**

**│ │**

**│ ├── 📂 services # Business Logic (Parsing, Ranking)**

**│ │ ├── ResumeParser.java (Interface)**

**│ │ ├── PdfResumeParser.java (Implements ResumeParser)**

**│ │ ├── DocResumeParser.java (Implements ResumeParser)**

**│ │ ├── ResumeParserFactory.java**

**│ │ ├── ResumeRanker.java**

**│ │ ├── ResumeProcessor.java**

**│ │**

**│ ├── 📂 utils # Helper Functions (Regex, File Handling)**

**│ │ ├── FileUtils.java**

**│ │ ├── TextExtractor.java**

**│ │**

**│ ├── 📂 ui # User Interface (CLI/Swing)**

**│ │ ├── ResumeCLI.java**

**│ │ ├── ResumeGUI.java**

**│ │**

**│ ├── 📂 main # Entry Point**

**│ │ ├── ResumeAnalyzerApp.java**

**│**

**├── 📂 resumes # Folder to store uploaded resumes**

**├── 📄 config.json # Configuration file (skills, keywords)**

**├── 📄 README.md # Project Documentation**

**├── 📄 pom.xml (if using Maven) / build.gradle (if using Gradle)**

**Core Modules & Blueprint**

**1️⃣ Resume Upload & Parsing**

* **Reads resumes and extracts relevant data**
* **Uses Java I/O for file handling**
* **Uses Factory Pattern to handle different file formats**
* **Implements Interface & Abstract Classes to define a common structure**

**Modules:**

* **ResumeParser (Interface) → Defines parsing method**
* **AbstractResumeParser (Abstract Class) → Provides base implementation**
* **PdfResumeParser → Extracts text from PDF resumes (implements ResumeParser)**
* **DocResumeParser → Extracts text from Word resumes (implements ResumeParser)**
* **TextExtractor → Uses Regex & String operations to extract structured details**

**2️⃣ Resume Data Model**

* **Stores extracted resume details**
* **Uses Encapsulation to store candidate details**
* **Implements Inheritance in different resume models if needed**

**Attributes:**

* **Name, Email, Skills, Experience, Education**

**3️⃣ Resume Ranking**

* **Compares extracted skills with required skills**
* **Uses Java Collections (List, Comparator) for sorting**

**Ranking Algorithm:**

* **+10 points for each matching skill**
* **+5 points per year of experience**

**4️⃣ Multithreading for Performance**

* **Uses ExecutorService (Thread Pool) to process multiple resumes in parallel**

**5️⃣ User Interface (CLI / GUI)**

* **CLI: Displays top-ranked resumes in console**
* **GUI (Swing/JavaFX): Displays ranked resumes in a table format**

**Advanced Features (Optional Enhancements)**

| **Feature** | **Concept Used** |
| --- | --- |
| **Database Storage (MySQL)** | **JDBC Integration** |
| **Web UI (Spring Boot)** | **REST APIs, Web UI** |
| **AI-based Skill Matching** | **NLP (Natural Language Processing)** |
| **Email Notification** | **Java Mail API** |

**Testing & Deployment**

* **Unit Testing: Use JUnit for testing parsing & ranking logic**
* **Logging & Debugging: Use Log4j for debugging errors**
* **Deployment: Package as JAR or deploy as Spring Boot Web App**

**Learning Outcomes**

| **Concept** | **Where Used?** |
| --- | --- |
| **OOP (Encapsulation, Inheritance, Polymorphism, Abstraction)** | **Candidate, Resume, AbstractResumeParser, ResumeParser Interface** |
| **Interfaces & Abstract Classes** | **Defining Resume Parsing structure** |
| **File Handling (Java I/O)** | **Reading resumes from disk** |
| **Exception Handling** | **Handling missing/corrupt files** |
| **Collections (List, Set, Map, Comparator)** | **Storing & ranking resumes** |
| **Multithreading (ExecutorService)** | **Processing multiple resumes in parallel** |
| **Comparator Interface** | **Sorting resumes based on score** |
| **Design Patterns (Factory, Singleton)** | **Factory for parsing, Singleton for config** |
| **Regex / String Manipulation** | **Extracting structured data from resumes** |

**Next Steps 🚀**

**1️⃣ Set up your Java project structure  
2️⃣ Implement Resume Parsing (PDF/DOCX) with Interface & Abstract Classes  
3️⃣ Add Resume Ranking logic using Comparator  
4️⃣ Optimize with Multithreading  
5️⃣ Choose CLI or GUI for output  
6️⃣ Enhance with AI, Database, or Web UI (Optional)**

**📌 Execution Flow Diagram**

plaintext

CopyEdit

[User Uploads Resume(s)]

⬇

[ResumeProcessor] (Handles File I/O & Parsing)

⬇

[ResumeParserFactory] (Decides PDF or DOC parser)

⬇

[ResumeParser Interface] ⬅ AbstractResumeParser (Base class)

⬇

[PdfResumeParser / DocResumeParser] (Extracts Data)

⬇

[Candidate Model] (Stores extracted details)

⬇

[ResumeRanker] (Compares with job skills & assigns score)

⬇

[Multithreading] (Optimizes processing for multiple resumes)

⬇

[CLI / GUI Output] (Displays top-ranked candidates)

**📂 File Responsibilities & Flow**

**1️⃣ Resume Upload & Processing**

📌 **Files:**

* ResumeProcessor.java → Handles file loading, validates input
* ResumeParserFactory.java → Uses **Factory Pattern** to decide parser

📌 **Flow:**  
✅ User uploads multiple resumes → ✅ Stored in /resumes/ folder  
✅ ResumeProcessor reads each file → ✅ ResumeParserFactory decides parser

**2️. Resume Parsing (OOP & Interfaces)**

📌 **Files:**

* ResumeParser.java → **(Interface)** Defines parseResume() method
* AbstractResumeParser.java → **(Abstract Class)** Common parsing logic
* PdfResumeParser.java → Implements parseResume() for **PDF files**
* DocResumeParser.java → Implements parseResume() for **DOCX files**

📌 **Flow:**  
✅ ResumeParserFactory calls PdfResumeParser or DocResumeParser  
✅ parseResume() extracts Name, Email, Skills, Experience, Education  
✅ Extracted data stored in Candidate.java

🛠 **Concepts Used:**  
✅ **Encapsulation & Inheritance** (Abstract class for common behavior)  
✅ **Polymorphism** (Interface allows different parsing implementations)

**3️⃣ Resume Data Model (OOP & Encapsulation)**

📌 **Files:**

* Candidate.java → Stores parsed resume details
* Resume.java → Holds additional metadata (file path, format)

📌 **Flow:**  
✅ Candidate objects store structured data like name, skills, experience

🛠 **Concepts Used:**  
✅ **Encapsulation** (Private fields, Getter/Setter methods)

**4️⃣ Resume Ranking & Scoring**

📌 **Files:**

* ResumeRanker.java → Compares resume skills with job requirements
* Comparator<Resume> → Sorts based on **matching skills & experience**

📌 **Flow:**  
✅ Extracted skills compared with config.json (Required skills)  
✅ Scores assigned:

* **+10 points per matching skill**
* **+5 points per year of experience**  
  ✅ Comparator sorts resumes **highest score first**

🛠 **Concepts Used:**  
✅ **Comparator Interface** (Custom sorting logic)  
✅ **Collections Framework (List, Set, Map)** (Stores & ranks resumes)

**5️⃣ Multithreading for Speed Optimization**

📌 **Files:**

* ResumeProcessor.java → Uses **ExecutorService (Thread Pool)**
* FutureTask → Asynchronous processing

📌 **Flow:**  
✅ Multiple resumes processed in parallel for efficiency

🛠 **Concepts Used:**  
✅ **Multithreading (Thread Pool, ExecutorService)**

**6️⃣ User Interface (CLI / GUI)**

📌 **Files:**

* ResumeCLI.java → **Command-line output** (Ranked list)
* ResumeGUI.java → **Graphical UI (Swing/JavaFX)**

📌 **Flow:**  
✅ ResumeRanker sends **sorted list of candidates** to UI  
✅ Display results in **table format (GUI) or text (CLI)**

🛠 **Concepts Used:**  
✅ **Java Swing/JavaFX** (For graphical display)  
✅ **MVC Architecture** (Separate UI from Business Logic)