

Goal

Design and implement an application for a leading job finder agency, allowing administrators to access and manage the agency's job dataset through an intuitive GUI. The application should enable the admin to view, add, update, and delete job listings, as well as filter, sort, and generate reports. Define the specific requirements for the GUI to ensure it provides efficient and user-friendly access to the dataset.

Introduction

You'll gain hands-on experience in designing and implementing a database for managing a job dataset. This includes setting up tables and relationships that match your data and implementing CRUD (Create, Read, Update, Delete) operations, enabling direct interaction with the database. The project emphasizes core OOP principles, with flexibility in designing classes that represent elements in your dataset. You'll implement features like UML diagrams, polymorphism, inheritance, and encapsulation, allowing you to build a well-structured and modular codebase. Building a GUI is essential for presenting and interacting with the data. Through this interface, users can view, filter, and manage data, while visual elements like charts help display trends and insights, making the application both functional and engaging.

Datasets

Job Dataset

A Comprehensive Job Dataset for Data Science, Research, and Analysis

From [Kaggle](#)

Technical tips

- **GUI:**
 - **Swing or JavaFX:** Use Swing or JavaFX to create a responsive and interactive interface, ensuring smooth navigation and functionality for the user.
- **Database:**
 - **JDBC:** Recommend JDBC for connecting to the database, enabling CRUD operations and other data management tasks.
 - **Database Choice (MySQL):** Use MySQL for a robust, server-based database if project scalability is important.
- **Class Diagram and Use Case Diagram:**
 - **draw.io:** Use draw.io to create clean and professional diagrams, including class and use case diagrams. Export diagrams as PDF or PNG for easy integration into documentation or reports. # Implementing details

Database

- Create a Database with Enhanced Functionality:
 - **Create:** Implement functionality to add new records to the database.
 - **Read:** Retrieve and display data from the database.
 - **Update:** Modify existing records within the database.
 - **Delete:** Remove records from the database as needed.
 - **Search and Filter:** Allow users to search for specific records or apply filters to refine the data shown.
 - **Sorting:** Enable sorting of records by key attributes to facilitate data organization and analysis.
 - **Data Export and Import:** Provide options to export data to formats CSV and

import data in the same form as the dataset.

GUI

- **Data Filtering and Search:**
 - Allow users to apply filters to view specific data, such as sorting by category, date, or other relevant attributes.
 - Implement search functionality to enable users to find specific records or data points quickly.
- **Presentation of Data:**
 - Design the interface to display data in a clear and organized manner, such as in tables, lists, or charts.
 - Utilize visual elements (e.g., graphs, charts) to help users interpret and analyze trends or patterns within the data.
- **Create and Export Reports:**
 - Enable the generation of summary reports using the data from the database.
 - Provide options for users to customize reports based on selected criteria, such as date ranges or specific data fields.
 - Allow users to export reports in formats like PDF or CSV for offline analysis and sharing.
- **CRUD Functionality & Database Manager:**
 - Implement buttons or form elements that allow users to Create, Read, Update, and Delete records directly from the GUI.
 - Build a database manager within the interface, allowing users to view, edit, sort, and manage database records, with changes reflected in real-time on the GUI.

OOP Principles & Java

- Demonstrate the four OOP principles (Polymorphism, Inheritance, Abstraction, Encapsulation).
- Use Built-in Collections:
 - Apply appropriate collection types across the program to manage data efficiently.
- Logging Framework:
 - Use a logging framework like Log4j or java.util.logging to track application events, errors, and performance metrics.
 - Maintain logs for ongoing maintenance, troubleshooting, and performance monitoring.
- Exception Handling:
 - Implement comprehensive exception handling to catch and manage errors gracefully.
 - Focus on handling database interaction errors (e.g., SQLExceptions) and user input issues to improve application stability.

Marking system & Requirements

%	Marking System	Requirements
(25)	Class diagram	<ul style="list-style-type: none">- (5) Create Class Diagram- (5) Relationships between Classes: Describe the associations, dependencies, or generalization relationships between classes.- (5) Methods & Attributes: Briefly explain the key methods and attributes in each class. Focus on relevant attributes and methods.
	Use Case diagram	<ul style="list-style-type: none">- (5) Create Use Case Diagram

%	Marking System	Requirements
(20)	OOP Principles	<p>- (5) Case Diagram: Discuss the main actor (Admin) and use cases (e.g., Create Job, Search Job, Export Report) shown in the diagram. Explain how the actor completes specific tasks through system interactions.</p> <p>- (5) Encapsulation: How do your classes remain modular while safeguarding data integrity?</p> <p>- (5) Abstraction: How did you design your class hierarchy to conceal unnecessary details and maintain clarity?</p> <p>- (5) Inheritance: How does your design allow a new class to inherit properties and methods from an existing class, organizing code into superclasses and subclasses?</p> <p>- (5) Polymorphism: How did you enable an object to assume multiple forms, especially through parent class references interacting with child class objects?</p>
(7.5)	Java Collections	<p>Explain how these collections are managed within classes to store and manipulate data</p>
(2.5)	Exception Handling and Logging	<p>Outline how you implemented exception handling and logging within the application to ensure stability and troubleshoot errors.</p>
(25)	Database	<p>- (10) CRUD functionality</p> <p>- (5) Search and Filter: Enable users to quickly locate specific records by entering search terms or applying filters based on various attributes, such as job title, category, or location. This feature helps users focus on relevant data and improves usability.</p> <p>- (5) Sorting: Allow users to sort records by key attributes making it easier to organize and analyze data in a meaningful way. Sorting options should support both ascending and descending order.</p> <p>- (5) Data Export and</p>

%	Marking System	Requirements
(20)	GUI	<p>Import/Offline functionality to export data in CSV format for easy sharing and offline analysis. Users can also import CSV files that match the dataset's structure to quickly populate or update the database with new records.</p> <p>- (10) Specify and build essential GUI features that meet the application's requirements, including interactive components for CRUD operations, search and filter options, and report generation.</p> <p>- (10) Data Display and Navigation: Design the GUI to display job data clearly, with options for tables, lists, or charts. Implement navigation components that allow users to easily access different functions, such as tabs for various features or panels for detailed views.</p>

Tips

- Begin by designing the GUI (without implementing it yet) and then continue with the class diagram.
- For OOP, it's good to start by implementing the basic structure of all classes, then define the relationships, and finally add the details.
- Have a backup plan to demonstrate requirements if the GUI doesn't work.