Abstract

KAAZ-ER-ADDA is a simple desktop app built with Java. Its main goal is to help freelancers find jobs that match their skills. In the real world, finding the right person for a job takes a lot of time because you have to read many profiles manually. This project solves that problem by using Artificial Intelligence (AI). The app lets users add freelancers and jobs into a list. Then, it sends this list to Google's Gemini AI. The AI reads the skills and the job needs, understands them, and tells us who is the best fit for which job. This project shows how we can use Java programming and Internet connections to make a smart tool that saves time.

1 Introduction and Project Overview

1.1 Purpose and Theme

The theme of the project is ”Smart Work.” The purpose is to make a tool that connects workers with jobs automatically. Instead of a human reading every CV, the computer does it for us.

1.2 Motivation

Why did I make this? Simple computer searches are not smart enough. For example, if a job needs ”ReactJS” but a freelancer writes ”Frontend Dev,” a normal computer search might say ”No match.” But an AI knows that ReactJS *is* a frontend tool. I wanted to build a system that understands these meanings using the Object-Oriented Programming (OOP) I learned in class.

1.3 Objectives

- To build a window (GUI) where users can type names and skills easily.
- To use OOP concepts like Inheritance and Polymorphism to write clean code.
- To connect the Java app to the internet (Google Gemini API).
- To match skills accurately, even if the words are slightly different.

1.4 Target Users

- **Small Business Owners:** Who need to hire someone quickly.
- **Freelancers:** Who want to see what jobs fit them.
- **Recruiters:** Who have too many resumes to read.

2 Development Process and Implementation

2.1 Tools and Technologies

- **Language:** Java (it is strong and reliable).
- **Design:** Java Swing (for buttons and text boxes).
- **Brain:** Google Gemini API (for the AI logic).
- **Connection:** `HttpURLConnection` (to send data to Google).

2.2 Key Classes and Code Explanations

2.2.1 1. The Parent Class (Abstraction)

I created a main abstract class called `NameOnly`. Both a ”Freelancer” and a ”Job” have names, so they both share this class. I made `getDetails()` abstract because a Job details look different from a Freelancer details.

```java
abstract class NameOnly {
    private String name;

    public NameOnly(String name){
        this.name = name;
    }
    // We don’t know how to get details yet,
    // so we leave this abstract for the children to decide.
    public abstract String getDetails();
}
```

Listing 1: The Abstract Parent Class

2.2.2 2. The Child Class (Inheritance)

The `Freelancer` class extends `NameOnly`. It uses the parent’s name storage but adds its own ”skill” variable.

```java
class Freelancer extends NameOnly{
    private String skill;

    public Freelancer(String name, String skill){
        super(name); // Sends name to parent class
        this.skill = skill;
    }

    @Override
    public String getDetails(){
        return "Freelancer: " + getName() + " with " + skill + " skill";
    }
}
```

Listing 2: The Freelancer Class

2.2.3 3. Managing the Data (Polymorphism)

This is the most important part. I used an `ArrayList` that holds `NameOnly` objects. Because of polymorphism, this single list can hold BOTH Freelancers and Jobs at the same time.

```java
// This list can hold any child of NameOnly
ArrayList<NameOnly> database = new ArrayList<>();

// Adding a Freelancer
database.add(new Freelancer(tName.getText(), tSkill.getText()));

// Adding a Job
database.add(new Job(tName.getText(), tSkill.getText()));
```

Listing 3: Storing different objects in one list

2.2.4 4. Connecting to AI

When the user clicks ”Assign Job”, the code creates a prompt and sends it to Google.

```java
// Create the connection
URI uri = new URI("https://generativelanguage.googleapis.com/...KEY");
URL url = uri.toURL();
HttpURLConnection con = (HttpURLConnection)url.openConnection();

// Tell the internet we are sending data (POST)
con.setRequestMethod("POST");
con.setDoOutput(true);

// Send the JSON (Data)
con.getOutputStream().write(json.getBytes("UTF-8"));
```

Listing 4: Sending data to Gemini AI

2.3 Class Diagram

This diagram shows how the classes are connected. You can see that `Freelancer` and `Job` are children of `NameOnly`, and the `Frame` uses the list.

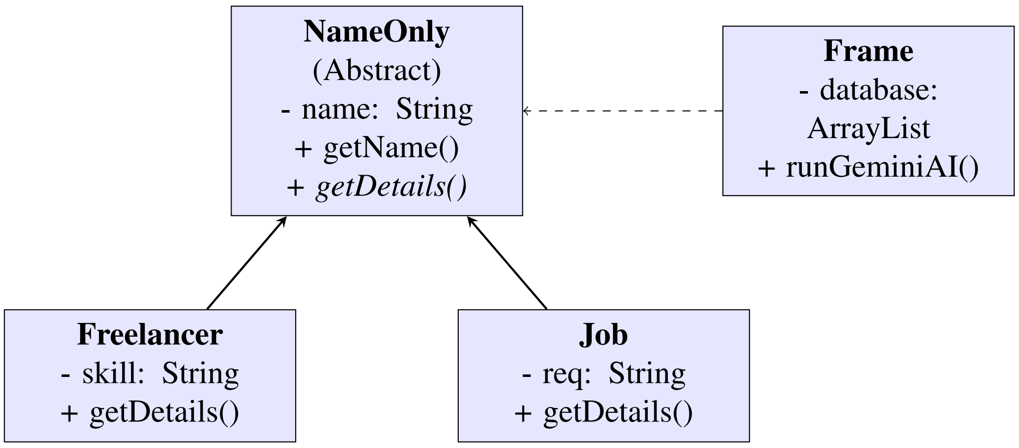


Figure 1: Visual Class Hierarchy showing Inheritance

2.4 OOP Principles Used

- **Encapsulation:** I used `private` variables (like name and skill) so they cannot be broken from outside. I used `getName()` to access them safely.
- **Inheritance:** `Freelancer` and `Job` take features from the `NameOnly` class. This saved me from writing the same code twice.
- **Polymorphism:** The list `database` treats everyone as `NameOnly`, but when the program runs, it knows exactly who is a Job and who is a Freelancer.

2.5 Challenges Faced

- **Reading the Answer:** Google sends the answer back in a complex format called JSON. Since I didn't use external libraries, extracting just the text answer using String code was tricky.
- **Prompt Engineering:** Telling the AI exactly what to do. At first, it gave long explanations. I had to tell it to ”Return a clean list” to get a simple output.

3 GitHub Repository

The complete source code for this project is available online at:

- **Project Link:** KAAZ-ER-ADDA GitHub Repository(clickable)

4 Installation

To run this project on your computer, follow these simple steps:

4.1 Prerequisites

1. You must have Java installed (JDK 8 or newer).
2. You must have an Internet Connection (because the AI lives in the cloud).

4.2 How to Run

1. Download the file named `Frame.java`.
2. Open your Command Prompt (cmd).
3. Go to the folder where you saved the file.
4. Type this to compile: `javac Frame.java`
5. Type this to run: `java Frame`

5 Results

5.1 The Main Screen

When you start the app, you see a clean window. You can type a name and a skill.

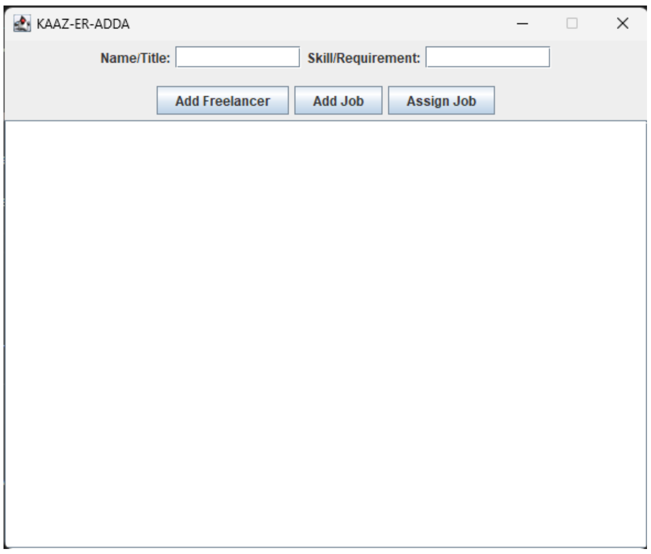
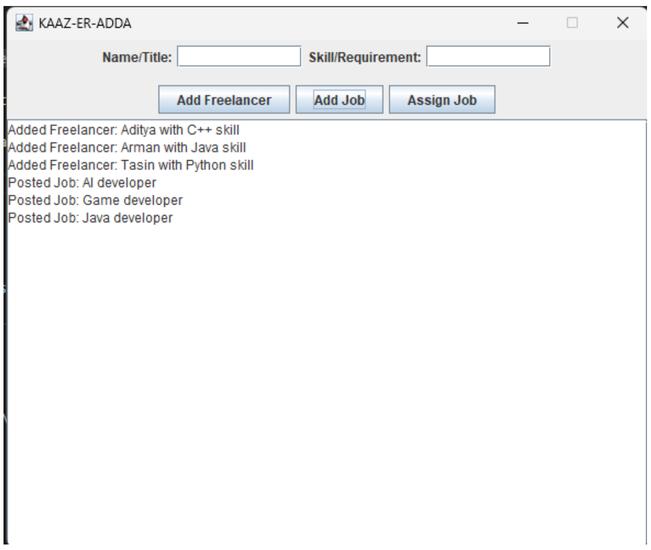


Figure 2: The KAAZ-ER-ADDA Interface

5.2 Adding People and Jobs

I added a freelancer named ”Arman” with ”Java” skills, ”Aditya” with ”c++” skills, ”Tasin” with ”python” skills, and a Job requiring ”Java developer”, ”Game developer” and ”AI developer”. The output box shows they were added successfully.

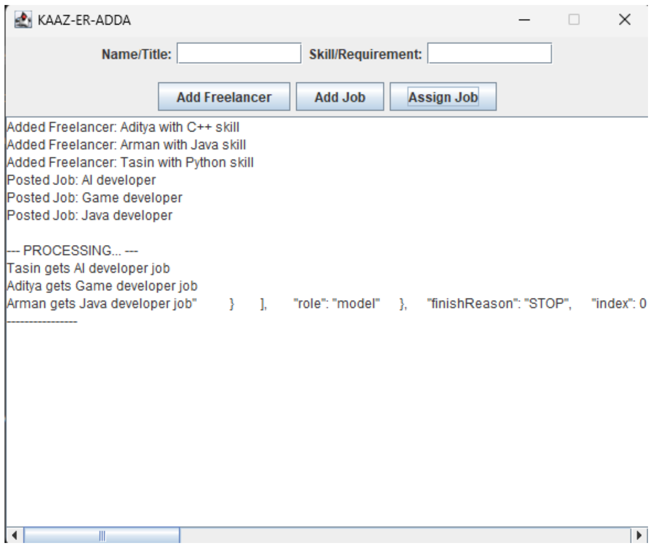


```
Added Freelancer: Aditya with C++ skill
Added Freelancer: Arman with Java skill
Added Freelancer: Tasin with Python skill
Posted Job: AI developer
Posted Job: Game developer
Posted Job: Java developer
```

Figure 3: Adding data to the system

5.3 The Final Match

After clicking ”Assign Job”, the AI correctly realized that ”Java” is good for ”Java developer”, ”python” is good for ”AI developer” and ”C++” is good for ”Game developer” matched them up!



```
Added Freelancer: Aditya with C++ skill
Added Freelancer: Arman with Java skill
Added Freelancer: Tasin with Python skill
Posted Job: AI developer
Posted Job: Game developer
Posted Job: Java developer

--- PROCESSING... ---
Tasin gets AI developer job
Aditya gets Game developer job
Arman gets Java developer job"      }      ],      "role": "model"     },     "finishReason": "STOP",     "index": 0
---------------
```

Figure 4: The AI successfully matching the job

6 My Contributions

This was an individual project developed entirely by me.

- **Concept:** I came up with the idea of using AI for recruitment.
- **Coding:** I wrote all the Java classes (`NameOnly`, `Freelancer`, `Job`, and `Frame`).
- **AI Integration:** I figured out how to send the HTTP request to Google Gemini and display the result.
- **Testing:** I tested the app with different skills to make sure the AI understands them.

7 Future Work

I plan to improve this project in the future:

1. **Saving Data:** Right now, if I close the app, the data is gone. I want to add a database so the information is saved forever.
2. **Fix some Bugs:** Still there are some bug in the output section. I want to update this in future.
3. **Better Design:** I want to make the buttons and colors look more modern.