A logo for college computing

Description automatically generated

**Assessment Cover Page**

|  |  |
| --- | --- |
| *Student Full Name* | Thiago Alves Coelho |
| *Student Number* | 2024287 |
| *Module Title* | Algorithms & Constructs  Project Skills & Professionalism  Software Development Fundamentals |
| *Assessment Title* | System Modelling & Build |
| *Lecturer/Supervisor* | Ken Healy / Muhammad Iqbal / Neil Doyle |
| *Assessment Due Date* | 07/04/2025 |
| *Date of Submission* | 10/05/2025 |

**Use of AI Tools**

I have not used any AI tools or technologies in the preparation of this assessment.

**Declaration**

By submitting this assessment, I confirm that I have read the CCT policy on academic misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source.

I declare it to be my own work and that all material from third parties has been appropriately referenced.

I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

**INTRODUCTION:**

To demonstrate the structure and functionality of an organization, a program in **JAVA®** has been designed as a prototype, incorporating fundamental principles of object-oriented programming. The system is built around three core classes—**Manager, Department, and Employee**—each with at least three distinct variations to represent different roles within the organization. Additionally, the program features an interactive console menu that allows users to input and validate data while generating appropriate random information. Custom search and sorting algorithms are implemented to optimize data management and retrieval. By integrating these elements, the prototype provides a functional simulation of an organizational environment, enabling a deeper analysis of its essential dynamics.

**EXPLAINNING THE SORTING & SEARCHING ALGORITHMS CHOICE**

I have used for my project the algorithms below to construct my main code:

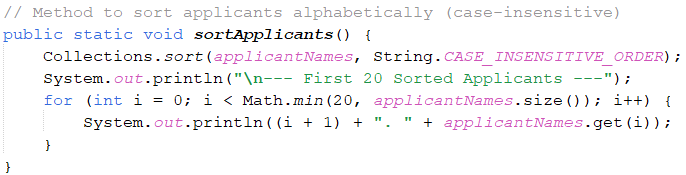
* **Sorting**: *Collections.sort(applicantNames, String.CASE\_INSENSITIVE\_ORDER)*
* **Searching**: a simple **linear search** inside a *for loop.*

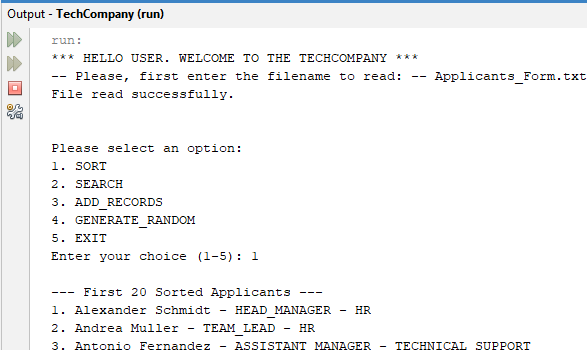
**WHY COLLECTIONS.SORT() IS A GOOD CHOICE:**

* **Efficiency for moderate-sized lists**:  
  *Collections.sort()* in Java is highly optimized internally — it uses a **Dual-Pivot Quicksort** *(optimized version of the traditional Quicksort)*for objects like Strings.  
  For lists with a few dozens or hundreds of elements, it is extremely **fast and efficient**.
* **Case-insensitive sorting**:  
  I sorted names ignoring uppercase/lowercase, thanks to *String.CASE\_INSENSITIVE\_ORDER*.  
  This ensures that "alice" and "Alice" for example, are treated the same — which makes the sorted list more natural and user-friendly.
* **Simplicity**:  
  Java's built-in sort is already **tested**, **optimized**, and **handles edge cases** (nulls, special characters) better than a manual sort.

**Alternative (and why not)**:

* I *could* implement my own **bubble sort**, **selection sort**, etc., but they are much **slower** and **more error-prone** for anything beyond tiny lists.
* *Collections.sort()* is based on much faster and professional algorithms under the hood.





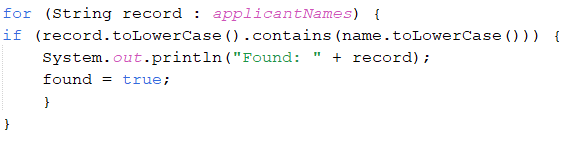
*Fig. 1 and 2 – Showing the main code for Sort and the output of the expected result in JAVA.*

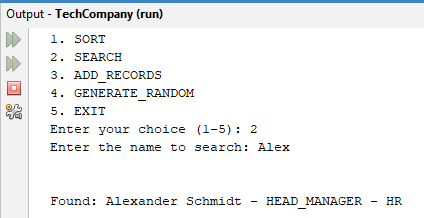
**WHY LINEAR SEARCH IS THE BEST FIT:**

* **List is unsorted or not purely names only**:  
  The list *applicantNames* contains **name + manager + department** (ex: "John Smith - IT\_MANAGER - DEVELOPMENT").  
  → Therefore, I cound't binary search it directly unless I isolated and sorted only names.  
  → **Linear search** (scanning one by one) is the safest method.
* **Flexibility**:  
  I did a **"contains"** match (record.toLowerCase().contains(name.toLowerCase())), not a strict match.  
  → A binary search **only works on fully sorted and strictly matching data**, so it would not handle partial matches like mine.
* **List size is small**:  
  My list of applicants is manageable (dozens, maybe hundreds).  
  For small lists, **linear search is just as fast**, and adding complexity with binary search would bring zero benefit.

**Alternative (and why not)**:

* **Binary search**: Only works on fully sorted data, and needs exact match. Would fail for "partial match" or if formatting varies.
* **HashMap lookup**: You could map names to employees, but that adds unnecessary memory and complexity for your project's size.





*Fig. 3 and 4 – Showing the main code for Search and the output of the expected result in JAVA.*

# LINK OF THE GITHUB PROJECT:

<https://github.com/thi-coelho/TechCompany.git>

# REFERENCES

GeeksforGeeks, 2025. *Collections.sort() in Java with Examples.* [Online]   
Available at: https://www.geeksforgeeks.org/collections-sort-java-examples/  
[Accessed 28 April 2025].

W3 Schools, 2025. *Java Enums.* [Online]   
Available at: https://www.w3schools.com/java/java\_enums.asp  
[Accessed 27 April 2025].

W3 Schools, 2025. *Java List Sorting.* [Online]   
Available at: https://www.w3schools.com/java/java\_sort\_list.asp  
[Accessed 28 April 2025].