**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | Algorithms & Constructs  Project Skills & Professionalism  Software Development Fundamentals |
| **Assessment Title:** | Integrated / Individual |
| **Lecturer Name:** | Ken Healy / Muhammad Iqbal / Neil Doyle |
| **Student Full Name:** | Orgilgerel Sumiya |
| **Student Number:** | 2024567 |
| **Assessment Due Date:** | 10th May 2025 |
| **Date of Submission:** |  |

**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. |

Table of Contents

[Algorithms & Constructs Report 3](#_Toc197737972)

[Recursive Sorting algorithm – MergeSort 3](#_Toc197737973)

[Searching algorithm – binarySearch 3](#_Toc197737974)

[Input validation and Menu System 3](#_Toc197737975)

[Generating random employees 3](#_Toc197737976)

[Code Structure and Quality 4](#_Toc197737977)

[Software Development Fundamentals: Modelling and Testing 5](#_Toc197737978)

[Use Cases 5](#_Toc197737979)

[Class Diagram 11](#_Toc197737980)

[Activity Diagram 13](#_Toc197737981)

[Sequence Diagram 15](#_Toc197737982)

[Comparison Between UML Models 16](#_Toc197737983)

[User Stories + Acceptance Criteria 16](#_Toc197737984)

# Algorithms & Constructs Report

## Recursive Sorting algorithm – MergeSort

For this project I used the merge sort algorithm in Sorter.java class to sort a list of employee names.

Merge sort is a method splits a list into 2 parts, sort each part, and then joins them back together in correct order. It is a recursive algorithm, meaning it calls itself again and again until list is sorted.

I chose merge sort because it is fast and works well even when the list is very long. It is a time complexity of O(n log n), which is better than bubble sort or insertion sort for large data. Merge sort keeps the original order of names if they are the same which is useful when sorting names. Quick sort wasn’t used because its wors-case performance is slower(O(n2)).

MergeSort method use a helper method called merge to join 2 sorted list. After sorting only first 20 names are show so output is easy to read.

## Searching algorithm – binarySearch

Search for an employee by name I use binary search in the Searcher.java class. Since list is already sorted with merge sort, binary search is good choice. It is very fast, with a time complexity of O(log n).

Binary search work by checking the middle of the lost. If the name is not there, it check either the left or right half and keeps going until it find the name or finish search. Program ignores uppercase and lowercase letters, so it’s easier for users to search.

If name is found , program show full details of that employee, including department and manger type. This help users get the information they need quickly.

## Input validation and Menu System

Program has a menu in terminal, built in Hospital.java class. It use an enum called MenuOption to control the different options. This make code more organized and easier to change later.

User can choose to sort, search , add a new employee , or create random employees. When adding an employee, user give a name and select things like department, job title , manager type, and position level. All inputs are checked to make sure they are correct. This help protect the data from error.

## Generating random employees

Program can a create random employees. This feature is in Hospital.java. It give each random employee a name like “emp77” and picks random values for department, manager type, job title, and position level using enum values. Random class use for this.

New employees are added to list and printed on screen along with any employees the user added. This is helpful for testing and reviewing full list.

## Code Structure and Quality

All code is inside the CA\_2 package. The classes hac=ve clear names and are short and easy to undersrand. Each class has comment to explain that it does. For example, Sorter.java and Searcher.java both have comments above method.

Menu use Enums, sorting uses recursion, and searching is done with custion binary search code. These features show a good understanding of algorithms and clean code. Program is easy to read, good-commented, and simple to maintain

# Software Development Fundamentals: Modelling and Testing

## Use Cases

A diagram of a user

AI-generated content may be incorrect.

Fig.1 Use Case Diagram

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | UC-01-AddEmployee | |
| **Description** (or Overview) | This use case allows the user to add a new employee to the hospital system by entering a name and selecting predefined options for department, job title, manager type, and position level. | |
| **Dependencies** | NONE | |
| **Precondition(s)** | System is running.  User is on the main menu screen | |
| **Primary Actor** | User(Admin or System Operator) | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction’>> | **Step** | **Action** |
| 1 | User selects ”Add Employee” from the menu |
| 2 | System prompts user to enter employee name |
| 3 | User enters name |
| 4 | System prompts for department, job title, position level, manager type |
| 5 | User selects values from predefined lists(enums) |
| 6 | System validates input |
| 7 | System creates an Employee object |
| 8 | Employee is added to list |
| 9 | System confirms addition |
| **Postcondition(s)** | The new employee is save in system.  Employee can now be searched, viewed or sorted. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 3 | **Action** |
| a | If user leaves name empty- system displays error and ask again |
| **Step 5** | **Action** |
| a | If user selects a value not in enum – system shows an error and re-prompts |
| **Comments (or Notes)** | This use case is implemented in Hospital.java, using addEmployee() method. Enums used ensure only valid values are entered | |

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | UC-02 ViewEmployees | |
| **Description** (or Overview) | This use case allow user to view all employees currently stored in the hospital system. List includes both manually added and randomly generated employees | |
| **Dependencies** | NONE | |
| **Precondition(s)** | System is running.  There is at least one employee in system. | |
| **Primary Actor** | User(Admin or System Operator) | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | User select “View Employees” from menu |
| 2 | System checks if there are any employees in the list |
| 3 | If employees exist, system displays details of each employee |
| 4 | If the list is empty , system show “No employees avialable” message |
| **Postcondition(s)** | User see a list of all employees stored in the system. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| a | If there are no employees in system, display message: “No employees avialable” |
| **Comments (or Notes)** | This use case is implemented in Hospital.java, using displayEmployees() method to loop through ArrayList<Employee>. | |

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | UC-03-SearchEmployee | |
| **Description** (or Overview) | This use case allow user to search for an employee by name. System use Binary search algorithm on a sorted list. If found , employee’s full details are displayed. | |
| **Dependencies** | Merge Sort (sorting must be done first) | |
| **Precondition(s)** | Employee list is sorted .  At least one employee exists. | |
| **Primary Actor** | User(Admin or System Operator) | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | User selects “Search Employee” from menu |
| 2 | System ask user to enter a name |
| 3 | User enters name |
| 4 | System performs binary search to find name |
| 5 | If found, system displays employee’s details |
| 6 | If not found, system show “Employee not found” message |
| **Postcondition(s)** | System either shows employee details or massage that the employee wasn’t found. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| a | If user enters an empty name – System displays error and asks again |
| **Step 4** | **Action** |
| a | If list is not sorted – system prompts user to sort list first |
| **Comments (or Notes)** | This use case is implemented in Searcher.java(binarySearch method) and used in Hospital.java | |

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | UC-04 GenerateRandomEmployee | |
| **Description** (or Overview) | This use case allow user to generate a random employee. System automatically assign a name(e.g “emp44”) and randomly selects values for department,job title, position level, and manager type | |
| **Dependencies** | NONE | |
| **Precondition(s)** | System is running.  User is on the main menu screen. | |
| **Primary Actor** | User(Admin or System Operator) | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | User selects “Generate Random Employee” from menu |
| 2 | System uses Random class to generate employee name (e.g emp77) |
| 3 | System randomly selects values for department, job title, manager type and position level from enums |
| 4 | System creates a new Employee object |
| 5 | Employee is added to list |
| 6 | System displays generated employee’s details |
| **Postcondition(s)** | New employee with randomly assigned attributes is added to system.  User can now view, sort or search for this employee. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| a | If system fails to generate random values – display error and retry |
| **Step 5** | **Action** |
| a | If list fails to update – system show error message and aborts addition |
| **Comments (or Notes)** | This use case is implemented in Hospital.java, using generateRandomEmployees() method. System use Random class and enums to ensure valid and varied entries. | |

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | UC-04 SortEmployees | |
| **Description** (or Overview) | This use case allows user to sort list of employees alphabetically by name using merge sort algorithm. Sorted list improves search accuracy and display order. | |
| **Dependencies** | Employee list must contain at least one entry | |
| **Precondition(s)** | System is running.  At least one employee exists in list | |
| **Primary Actor** | User(Admin or System Operator) | |
| **Main Sequence**  << describes the actions that take place in a ‘normal interaction>> | **Step** | **Action** |
| 1 | User selects “Sort Employee” from menu |
| 2 | System checks if list contains employees |
| 3 | System applies merge sort algorithm to employee list |
| 4 | List is sorted in ascending(A-Z) order by employee name |
| 5 | System displays first 20 sorted employee names for clarity |
| **Postcondition(s)** | Employee list is sorted alphabetically by name.  Sorted list is ready for search or display. | |
| **Alternative Paths** (or Exceptions)  <<sets out alternative sequence(s) of actions in unusual situations, such as a user error  If there are none then this section would be omitted >> | **Step** 2 | **Action** |
| a | If no employees exist – system displays:”No employees to sort” |
| **Step 3** | **Action** |
| a | If sorting fails – system shows error and returns to main menu |
| **Comments (or Notes)** | This use case is Implemented in Sorter.java using mergeSort() and merge() methods. It is called from the Hospital.java menu system when the sort option is selected. | |

## Class Diagram

A diagram of a company

AI-generated content may be incorrect.

Fig.2 Class Diagram

**Explanation**

Class Diagram above represents internal structure of the Hospital Management System. It show how classes and enums are related and how data flows between them.

* Employee is core class, representing individual staff members in system. It contains attributes such **as name** and **positionLevel** and includes a method **getDetails()** to return formatted employee information.
* ManagerType and Department are modelled as enums. These provide predefined options for type of manager(e.g GENERAL, SPECIALIZED) and department(e.g HR, FINANCE,MEDICAL), ensuring consistent data entry.
* **HOSPITAL** class is responsible for managing all employees. It has a list of **employees(List<Employee>)** and **a Random** object to generate random entries. It contains methods like **addEmployee(Employee)** and **viewEmployees()** to interact with system.

Diagram highlights that **Hospital** has composition relationship with **Employee**, meaning that employees are tightly coupled with hospital system. Use of enums also simplifies data validation and logic in application.

**Justification for Choosing Class Diagram**

Class Diagram was chosen because it is most effective way to:

* Visually represent the **object-oriented structure** of the system.
* Show relationships between core components like **Employee, Hospital**  and enums.
* Support documentation and communication during software development.

Compared to other diagrams such as Object or Component diagrams, Class Diagram gives a **high-level overview of system design**, making it easier to plan, code and debug.

For a system implemented in java with enums and collections, this model gives most accurate technical structure.

## Activity Diagram

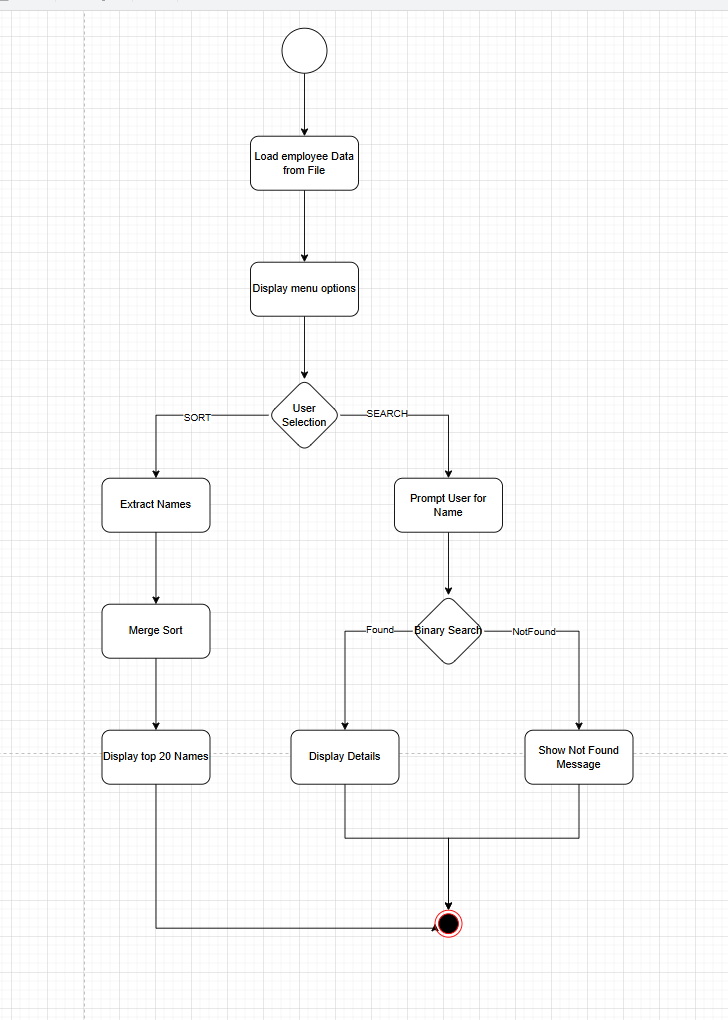


Fig.3 activity Diagram

**Explanation**

The Activity Diagram shows what happens when the user starts program. First, system loads employee data and shows a menu. Then user chooses an action like sorting or searching.

If user selects “Sort”, system asks for a name and runs a binary search. If name is found , it shows details. If not, it shows a “Not Found” message.

This diagram helps understand step by step process and how system makes decisions based on user’s choice.

**Justification for Choosing Activity Diagram**

I chose Activity Diagram because it clearly shows steps and decisions in hospital system. It is easy to see how user moves from one step to another .

It is useful for showing actions like sorting or searching and for showing how system reacts to different inputs. This make diagram helpful for both developers and non technical people.

Compared to State Diagram or Sequence Diagram, Activity Diagram is better for showing whole process that is why it is best choice describe how the system works

## Sequence Diagram

A diagram of a search for an employee

AI-generated content may be incorrect.

Fig.4 Sequence Diagram

**Explanation**

Sequence Diagram show how the user searches for an employee in hospital system. First, user selects “Search Employee” option from menu. System then asks user to enter a name.

After receiving name, Hospital class calls Sorter class to sort emplotee list using merge sort. Then it sends sorted list to Searcher class to run a binary search.

Searcher checks list and returns either employee details or a message saying employee wasn’t found. Finally, Hospital class shows the result to user.

This diagram helps to understand how different parts of system work together step by step during a search action.

**Justification for Choosing Sequence Diagram**

I chose Sequence Diagram because it shows flow of messages between different parts of system during a specific action. It is useful for understanding timing and order of method calls.

In this case, system uses more than one class – Hospital, Sorter, Searcher and EmployeeList to complete search. Diagram clearly shows how these classes interact.

Compared to class or Activity Diagrams, Sequence Diagram gives a better view of object communication. It helps developers understand how system behaves during one user task.

## Comparison Between UML Models

Each UML diagram used in project shows a different part of system.

**Class Diagram** shows structure of program. It shows main classes, attributes and relationships. This helps when designing code and understanding how objects are related.

**Activity Diagram** shows steps and actions taken by user and system. It is useful for understanding how system behave in different situations, like adding or searching for employees.

**Sequence Diagram** shows how objects talk to each other during one task. It explains order of actions and messages between classes. This is helpful to follow what happens when user runs a specific function.

Compared to each other, all three diagrams help in different ways:

* Class Diagram is best for structure.
* Activity Diagram is best for flow and decision making.
* Sequence Diagram is best for message and method call order.

Together, they give a full picture of how the system works from design and action point of view

## User Stories + Acceptance Criteria

**1.Add Employee**

As a system operator, I want to add a new employee, so that they are saved in hospital system.

Acceptance Criteria:

-The user must enter a valid name.

-The user must select department, manager type, and position level from given options.

-The system should confirm that the employee is added.

**2.View Employees**

As a user, I want to see all current employees, so that I know who is in the system.

Acceptance Criteria:

-If the list is not empty, all employee details are shown.

-If no employees exist, a message “No Employees available” is shown.

**3.Search Employee**

As a user, I want to search for an employee by name, so that I can view their details.

Acceptance Criteria:

-The user must enter a name.

-If found, the system displays the employee’s department and manager type.

-If not found, the system shows “Employee not found”.

**4.Generate Random Employee**

As a system operator, I want to generate random employees, so that I can test the system easily.

Acceptance Criteria:

-Random name must be generated using a format like ”emp77”.

-Random values must be chosen from enum lists.

-New employees must be added and shown in the list.

**5.Sort Employees**

As a user, I want to sort employees by name, so that it is easier to find someone.

Acceptance Criteria:

-Sorting uses merge sort to arrange names A-Z.

-Only the first 20 names are shown after sorting.

-The system must show a message if there are no employees.

## Unit Test

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **001** | **Developer:** | **Orgilgerel** | **Date Test Carried Out:** | **2025-05-10** | **Test Name:** | | **Add Valid Employee** | |
| **Module Tested** | *Hospital.Java* | | |  | | | | | |
| **Description of Test** | *Check if a new employee is added correctly when valid details are entered* | | | **Test Carried out by:** | *Orgilgerel* | |  | | |
| **Test Precondition(s)** | System is running and on main menu | | | | | | | | |
| **Dependencies (if any)** | Enums and employee list initialized | | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | | **PASS/FAIL** | **NOTES** |
| 1 | Name = “Alice”, Department = HR, ManagerType = General | | Employee is added to list | | Employee “Alice” was added successfully. | | | PASS | AddEmployee method in Hospital.java used |
|  |  | |  | |  | | |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **002** | **Developer:** | **Orgilgerel** | **Date Test Carried Out:** | **2025-05-10** | **Test Name:** | | **Add Valid Employee** | |
| **Module Tested** | *Hospital.Java* | | |  | | | | | |
| **Description of Test** | *Check if the system correctly handles empty name input.* | | | **Test Carried out by:** | *Orgilgerel* | |  | | |
| **Test Precondition(s)** | System is running and on main menu | | | | | | | | |
| **Dependencies (if any)** | Enums and employee list initialized | | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | | **PASS/FAIL** | **NOTES** |
| 1 | Name = “”, Department = Finance, ManagerType = TeamLead | | System shows error message and asks again | | Error: Name cannot be empty | | | PASS | Name validation logic tested |
|  |  | |  | |  | | |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **003** | **Developer:** | **Orgilgerel** | **Date Test Carried Out:** | **2025-05-10** | **Test Name:** | | **Search for existing employee** | |
| **Module Tested** | *Searcher.Java* | | |  | | | | | |
| **Description of Test** | *Check if the system finds and displays correct employee details.* | | | **Test Carried out by:** | *Orgilgerel* | |  | | |
| **Test Precondition(s)** | System is running and on main menu | | | | | | | | |
| **Dependencies (if any)** | Enums and employee list initialized | | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | | **PASS/FAIL** | **NOTES** |
| 1 | Search Name = “Alice” | | Employee details are shown | | Employee “Alice” was found and details displayed | | | PASS | binarySearch method tested |
|  |  | |  | |  | | |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **004** | **Developer:** | **Orgilgerel** | **Date Test Carried Out:** | **2025-05-10** | **Test Name:** | | **Seartch for non-existing employee** | |
| **Module Tested** | *Searcher.Java* | | |  | | | | | |
| **Description of Test** | *Check if system shows a message when name is not found* | | | **Test Carried out by:** | *Orgilgerel* | |  | | |
| **Test Precondition(s)** | System is running and on main menu | | | | | | | | |
| **Dependencies (if any)** | Enums and employee list initialized | | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | | **PASS/FAIL** | **NOTES** |
| 1 | Search Name = “Bob” | | System shows “Employee not found” | | System displayed: “Employee no found” | | | PASS | Error handling for failed search |
|  |  | |  | |  | | |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TEST ID NUMBER** | **005** | **Developer:** | **Orgilgerel** | **Date Test Carried Out:** | **2025-05-10** | **Test Name:** | | **Generate random employee** | |
| **Module Tested** | *Hospital.Java* | | |  | | | | | |
| **Description of Test** | *Check if the system generates a random employee with valid attributes.* | | | **Test Carried out by:** | *Orgilgerel* | |  | | |
| **Test Precondition(s)** | System is running and on main menu | | | | | | | | |
| **Dependencies (if any)** | Enums and employee list initialized | | | | | | | | |
| **TEST STEP** | **DATA (Input)** | | **Expected Result(s)** | | **Actual Result(s)** | | | **PASS/FAIL** | **NOTES** |
| 1 | Trigger random generation | | Random employee is added and displayed | | Random employee”emp55” was added and displayed with valid attributes | | | PASS | Error handling for failed search |
|  |  | |  | |  | | |  |  |