Even Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Data Structure Final Project

(Group Work)

Student Information:

	Surname:	**Given Name:**	**Student ID:**
1.	Keisha	Edelyne	2602169850
2.	Pandiora	Felise Amore	2602174453

Course Code : COMP6048001 **Course Name :** Data Structures

Class : L2BC **Lecturer :** 1. Dr. Maria Seraphina Astriani, S.Kom., M.T.I.
2. Nunung Nurul Qomariyah, S.Kom., M.T.I., Ph.D.

Type of Assignment : Final Project Report

Submission Pattern :

Due Date : 20 June 2023 **Submission Date :** 20 June 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offences which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Edelyne Keisha
Felise Amore Pandiora

Table of Contents

Introduction

Background

In today's rapidly advancing technological era, people are leaving behind old methods and embracing the power of technology and the Internet to enhance their productivity. This change can be seen in the business world, where businesses of all sizes, whether big or small, no longer need to use physical books and handwritten notes. Instead, they are starting to use online management systems, which offer many tools to make jobs easier and ensure the business can go on smoothly. The introduction of these online tools has changed how businesses operate, simplifying the process of record-keeping and making it easier and faster to access crucial information. As a result, businesses can keep up in a fast-paced and competitive market.

However, not everyone can keep up with the rapidly changing era, and not everyone comes equipped with the skills to do so. In this case, it becomes difficult to utilise management systems that strike a balance between accessibility and efficiency. On one hand, some applications offer a lot of advanced functions. However, they require a deep learning curve that most cannot keep up with. On the other hand, applications that are simple and easy to use often do not have enough necessary features, which hinders the business from fully harnessing the benefits of modern technology.

Problem Description

This is the report for our data structures final project, which we made to put our knowledge of having explored different data structures and how they are implemented throughout this semester into practice and evaluate how these data structures perform in real-life situations. Choosing the right data structure is crucial, especially when working with substantial amounts of data. Even a slight variation in efficiency can make a substantial difference in the overall performance of a system.

Many business owners find it hard and time-consuming to keep track of every single one of their employees, and it is harder to look for one specific employee if they have thousands of them. Thus, we propose Employee Central, an employee management system to help them manage their employees.

In order to find the most suitable data structure for our program, Employee Central, we are making an analysis of running this employee management system using different data structures. Through this analysis, we can then observe which data structure is more efficient in terms of time complexity and space complexity.

Objective

Our main objective is to find the most efficient data structure for our Employee Management System. To do this, we will analyse the time complexity of these methods:

1. Add Employee
2. Remove Employee
3. Edit Employee
4. Search Employee

In addition to this, we will also compare the space complexity of the program to see how much space each method will take. With that data, we will find the most efficient data structure based on the speed and memory it takes to run.

Project Specifications

- Software and Library used:
  - Jetbrains IDE, Intellij IDEA
  - Java Development Kit (JDK)
  - java.io.File
  - java.util.ArrayList
  - java.util.Iterator
  - java.util.LinkedList
  - java.util.Queue
  - java.util.Map
  - java.util.TreeMap
  - java.util.Scanner
  - java.util.Random
  - java.lang.System.nanoTime()
- Input:
  - addEmp() method
  - ➔ asks the admin for employee details such as employee ID, name, phone number, address, department, position, gender, and email.
  - removeEmp() method
  - ➔ asks the admin for the employee ID, as the system removes employees and their details based on ID.
  - editEmp() method
  - ➔ asks the admin for the employee ID and other new details such as new name, new phone number, new address, new department, new position, new gender, and new email.
  - searchEmp() method
  - ➔ asks the admin for the employee ID, as the system searches for employee details based on ID.
- Output:
  - Add Employee
  - ➔ After inputting all details, the system will display "Employee has been added!"
  - Remove Employee
  - ➔ If the employee list is empty (no employees yet), the system will display "No employees at the moment!"
  - ➔ If the employee ID is not found (does not exist), the system will display "Employee with that ID does not exist!"

- ➔ After removing the employee details, the system will display “Employee has been removed!”
- ○ Edit Employee Data
- ➔ If the employee list is empty (no employees yet), the system will display “No employees at the moment!”
- ➔ If the employee ID is not found (does not exist), the system will display “Employee with that ID does not exist!”
- ➔ After updating employee details, the system will display “Employee data has been updated!”
- ○ View Employee List
- ➔ If the employee list is empty (no employees yet), the system will display “No employees at the moment!”
- ➔ If the employee list is not empty (there are available employee(s)), the system will display all employee and their details (ID, name, phone number, address, department, position, gender, and email)
- ○ Search Employee by ID
- ➔ If the employee list is empty (no employees yet), the system will display “No employees at the moment!”
- ➔ If the employee ID is not found (does not exist), the system will display “Employee with that ID does not exist!”
- ➔ If the employee ID is found, the system will display the details of that employee (name, phone number, address, department, position, gender, and email)
- ○ Exit System
- ➔ The system display “Thank you for using Employee Central. Hope to see you again☺”
- ➔ Quit the program (program ends)

➢ Alternative Data Structures:
- ○ ArrayList
- ○ HashMap
- ○ Queue
- ○ TreeMap

Analysis of Data Structures

Array List

The first data structure we tried to use was an Array List, as it is one of the more common data structures to use. Array List is a data structure that's more dynamic than a normal list, as we don't need to specify the size of the array, and can just add and delete as needed. It is used to store a collection of data. It is easy to search and modify in an Array List, as all we need to do is just search for its index.

However, it has a time complexity of O(N) because doing anything to the array requires the elements to shift based on the number of elements in the list.

It also has a space complexity of O(N) because it requires memory equal to the amount of elements in the list.

Queue

The second data structure we implemented is a Queue. It is a data structure that follows the First In, First Out principle. Essentially, this data structure would add the elements to the back of the list, and then remove the elements at the front, like a line in a queue. This ensures that we can manage data in a sequential order, because the elements are processed in the order that they are added.

For adding or deleting an element, it has a time complexity of O(1), since only the front element can be removed at a time and insertion can only take place at the back. However, for searching, it can have a time complexity of O(N) because reaching any specific element isn't possible without removing the elements stored after it.

On the other hand, the space complexity of a Queue is O(1), as no extra space is required for any operation.

Hash Map

The third data structure we used is a Hash Map. It's a data structure that focuses on the use of key-value pairs. It makes use of a technique called Hashing, where the keys are transformed into unique numerical indices. This allows us to look for the values faster because all we need to do is search for their keys.

The time complexity of a Hash Map is O(1) for insertion, deletion, and retrieval operations. However, in the worst case scenario, it would be O(N) because of collisions.

The space complexity of a HashMap is O(N), because with the increase of the number of entries, the hashmap's space will increase linearly.

Tree Map

The last data structure that we tried to implement is a Tree Map. It is a data structure that also utilises key-value pairs. However, the difference here is that it uses a balanced binary search tree, typically a red-black tree, while a Hashmap uses an array. Using a BST makes sure that there are no duplicates. It also sorts the key-values in a sorted order, so it allows for efficient operations that rely on sorted keys, such as finding the smallest or largest key or iterating over the elements in a specific order.

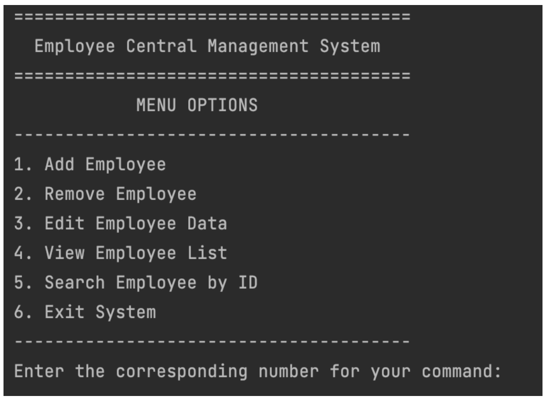
The time complexity of a Tree Map is O(log(N)), with N being the number of elements in the Tree Map, it is because of the nature of the tree.

The space complexity of a Tree Map is O(N) because the Tree Map needs to save memory to store the key-value pairs. The space required grows linearly with the number of elements in the TreeMap.

Screenshots of Working System

➔ All of the data structures (ArrayList, HashMap, Queue, and TreeMap) work similarly and display the exact same output as below:

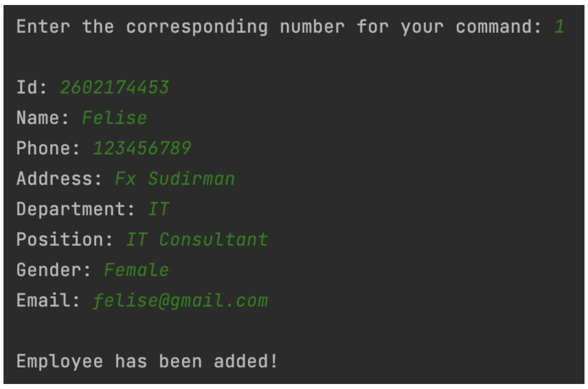
- Menu Display



```
======================================
  Employee Central Management System
======================================
            MENU OPTIONS
--------------------------------------
1. Add Employee
2. Remove Employee
3. Edit Employee Data
4. View Employee List
5. Search Employee by ID
6. Exit System
--------------------------------------
Enter the corresponding number for your command:
```

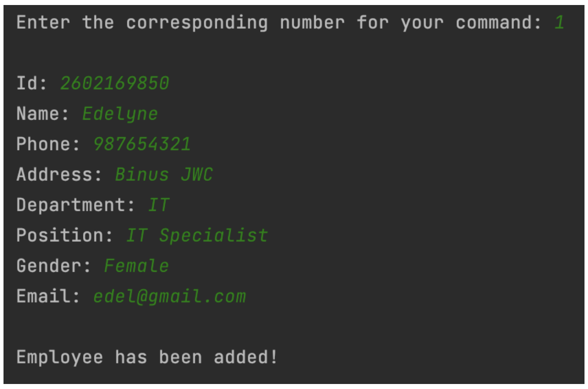
fig 1.1 Display Menu

- Add Employee



```
Enter the corresponding number for your command: 1

Id: 2602174453
Name: Felise
Phone: 123456789
Address: Fx Sudirman
Department: IT
Position: IT Consultant
Gender: Female
Email: felise@gmail.com

Employee has been added!
```

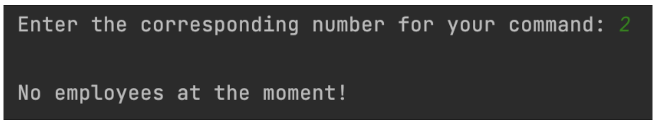
fig 1.2 Add Employee 1



```
Enter the corresponding number for your command: 1

Id: 2602169850
Name: Edelyne
Phone: 987654321
Address: Binus JWC
Department: IT
Position: IT Specialist
Gender: Female
Email: edel@gmail.com

Employee has been added!
```

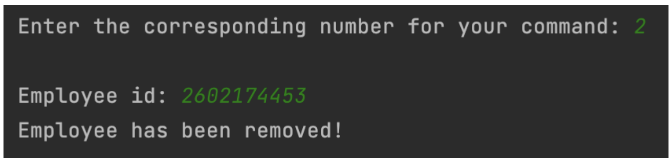
fig 1.3 Add Employee 2

- Remove Employee



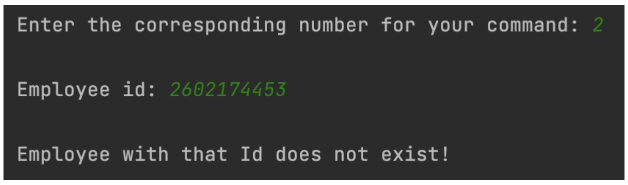
```
Enter the corresponding number for your command: 2

No employees at the moment!
```

fig 1.4 When employee list is empty



```
Enter the corresponding number for your command: 2

Employee id: 2602174453
Employee has been removed!
```

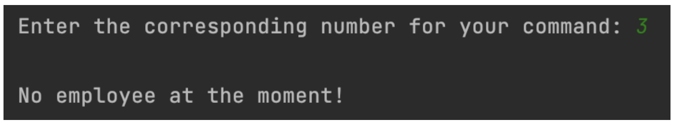
fig 1.5 Remove Employee



```
Enter the corresponding number for your command: 2

Employee id: 2602174453

Employee with that Id does not exist!
```

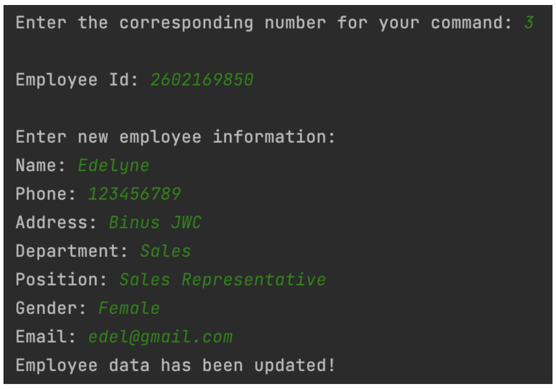
fig 1.6 ID not found

- Edit Employee



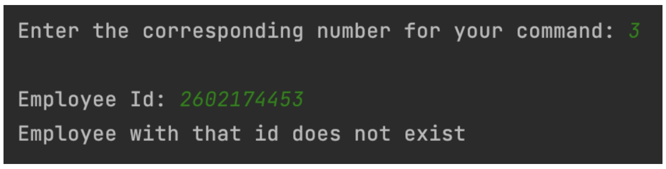
```
Enter the corresponding number for your command: 3

No employee at the moment!
```

fig 1.7 When employee list is empty



```
Enter the corresponding number for your command: 3

Employee Id: 2602169850

Enter new employee information:
Name: Edelyne
Phone: 123456789
Address: Binus JWC
Department: Sales
Position: Sales Representative
Gender: Female
Email: edel@gmail.com
Employee data has been updated!
```

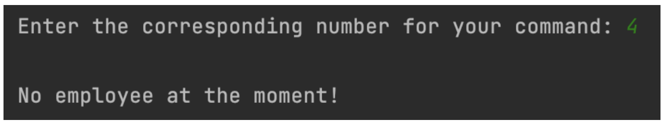
fig 1.8 Edit Employee



```
Enter the corresponding number for your command: 3

Employee Id: 2602174453
Employee with that id does not exist
```

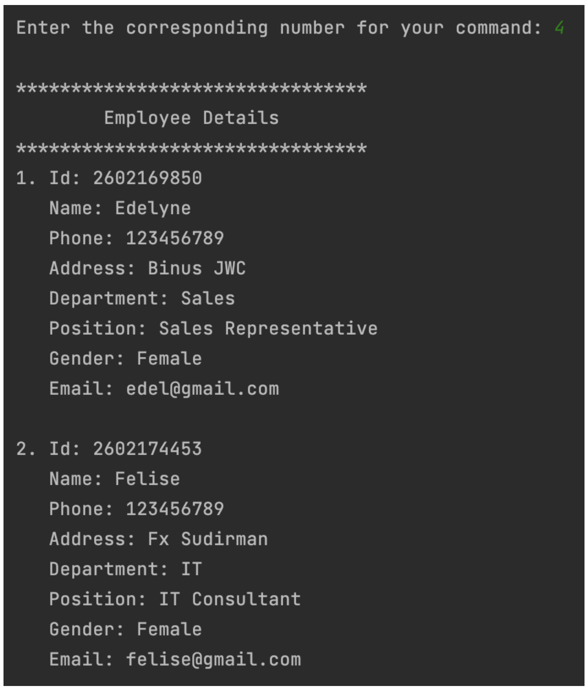
fig 1.9 ID not found

- View Employee List



```
Enter the corresponding number for your command: 4

No employee at the moment!
```

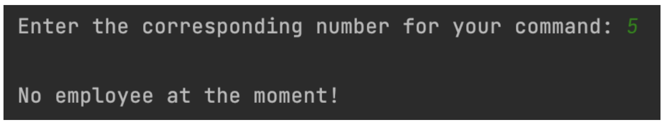
fig 1.10 When employee list is empty



```
Enter the corresponding number for your command: 4

********************************
        Employee Details
********************************
1. Id: 2602169850
   Name: Edelyne
   Phone: 123456789
   Address: Binus JWC
   Department: Sales
   Position: Sales Representative
   Gender: Female
   Email: edel@gmail.com

2. Id: 2602174453
   Name: Felise
   Phone: 123456789
   Address: Fx Sudirman
   Department: IT
   Position: IT Consultant
   Gender: Female
   Email: felise@gmail.com
```

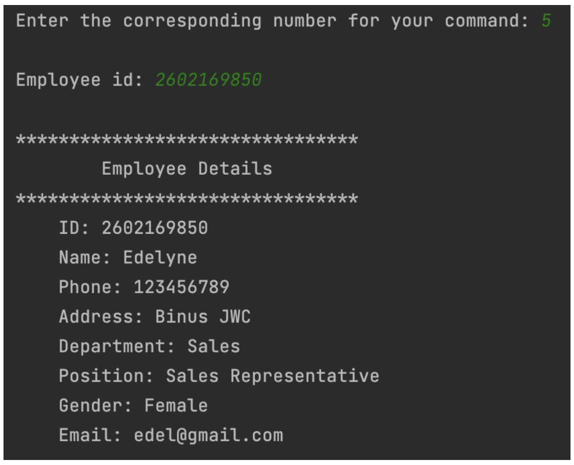
fig 1.11 View Employee List

- Search Employee by ID



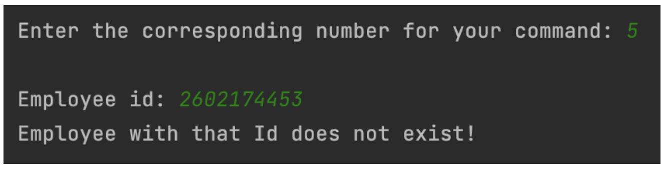
```
Enter the corresponding number for your command: 5

No employee at the moment!
```

fig 1.12 When employee list is empty



```
Enter the corresponding number for your command: 5

Employee id: 2602169850

********************************
        Employee Details
********************************
    ID: 2602169850
    Name: Edelyne
    Phone: 123456789
    Address: Binus JWC
    Department: Sales
    Position: Sales Representative
    Gender: Female
    Email: edel@gmail.com
```

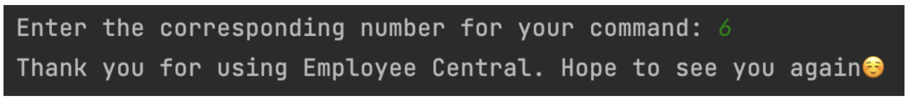
fig 1.13 Search Employee by ID



```
Enter the corresponding number for your command: 5

Employee id: 2602174453
Employee with that Id does not exist!
```

fig 1.14 ID not found

- Exit System

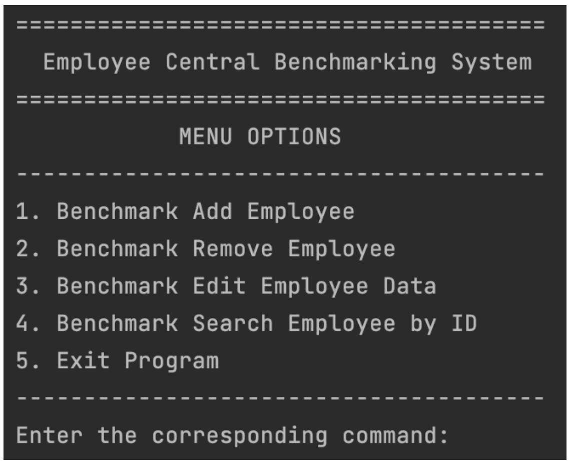


```
Enter the corresponding number for your command: 6
Thank you for using Employee Central. Hope to see you again☺
```

fig 1.15 Exit System

➔ The BenchmarkTest.java file works as below:

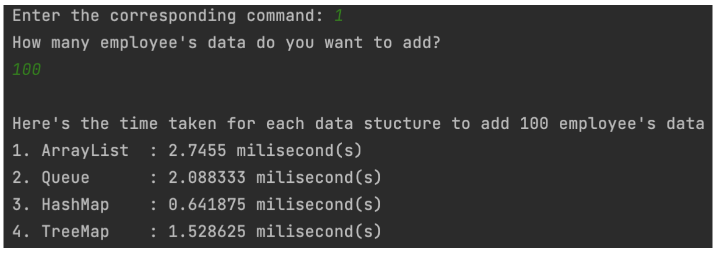
- Menu Display



```
=======================================
  Employee Central Benchmarking System
=======================================
             MENU OPTIONS
---------------------------------------
1. Benchmark Add Employee
2. Benchmark Remove Employee
3. Benchmark Edit Employee Data
4. Benchmark Search Employee by ID
5. Exit Program
---------------------------------------
Enter the corresponding command:
```

fig 2.1 Display Menu

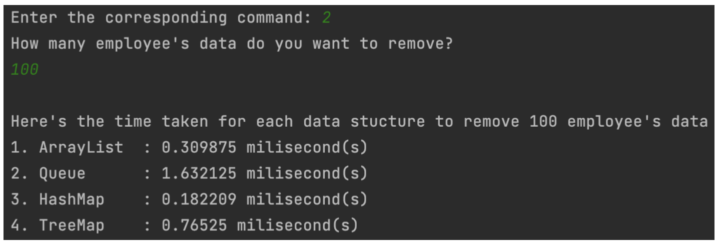
- Benchmark Add Employee



```
Enter the corresponding command: 1
How many employee's data do you want to add?
100

Here's the time taken for each data stucture to add 100 employee's data
1. ArrayList  : 2.7455 milisecond(s)
2. Queue      : 2.088333 milisecond(s)
3. HashMap    : 0.641875 milisecond(s)
4. TreeMap    : 1.528625 milisecond(s)
```

fig 2.2 Benchmark Adding 100 Employees

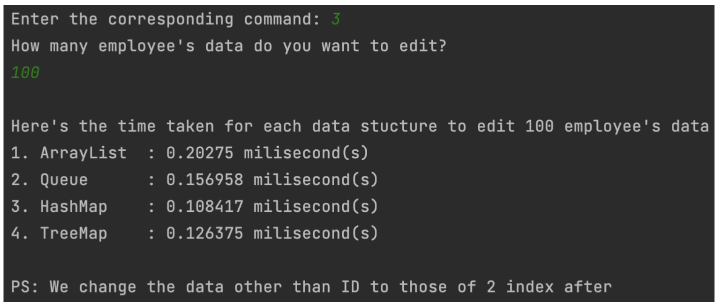
- Benchmark Remove Employee



```
Enter the corresponding command: 2
How many employee's data do you want to remove?
100

Here's the time taken for each data stucture to remove 100 employee's data
1. ArrayList  : 0.309875 milisecond(s)
2. Queue      : 1.632125 milisecond(s)
3. HashMap    : 0.182209 milisecond(s)
4. TreeMap    : 0.76525 milisecond(s)
```

fig 2.3 Benchmark Removing 100 Employees

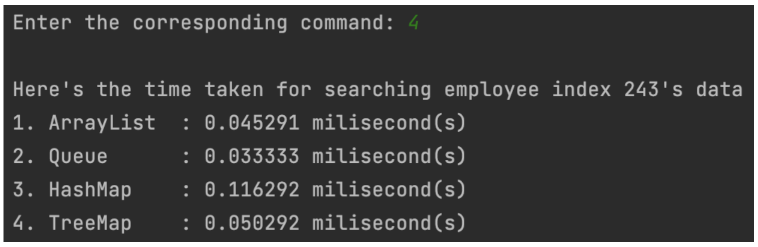
- Benchmark Edit Employee



```
Enter the corresponding command: 3
How many employee's data do you want to edit?
100

Here's the time taken for each data stucture to edit 100 employee's data
1. ArrayList  : 0.20275 milisecond(s)
2. Queue      : 0.156958 milisecond(s)
3. HashMap    : 0.108417 milisecond(s)
4. TreeMap    : 0.126375 milisecond(s)

PS: We change the data other than ID to those of 2 index after
```

fig 2.4 Benchmark Editing 100 Employees

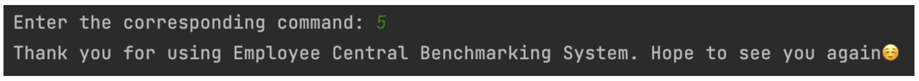
- Benchmark Search Employee by ID



```
Enter the corresponding command: 4

Here's the time taken for searching employee index 243's data
1. ArrayList  : 0.045291 milisecond(s)
2. Queue      : 0.033333 milisecond(s)
3. HashMap    : 0.116292 milisecond(s)
4. TreeMap    : 0.050292 milisecond(s)
```

fig 2.5 Benchmark Search Emp for random employee

- Exit Program

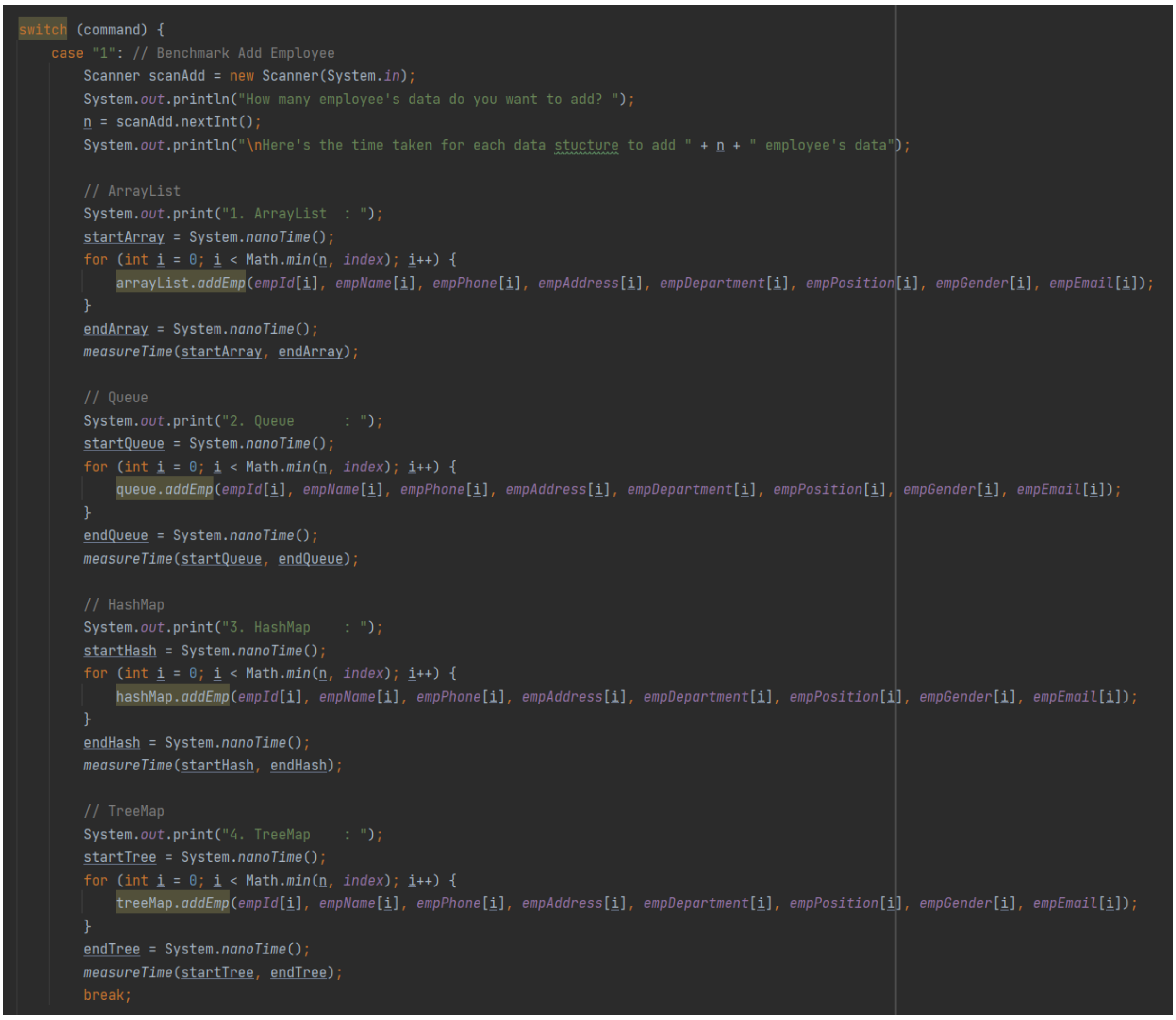


```
Enter the corresponding command: 5
Thank you for using Employee Central Benchmarking System. Hope to see you again😌
```

fig 2.6 Exit program

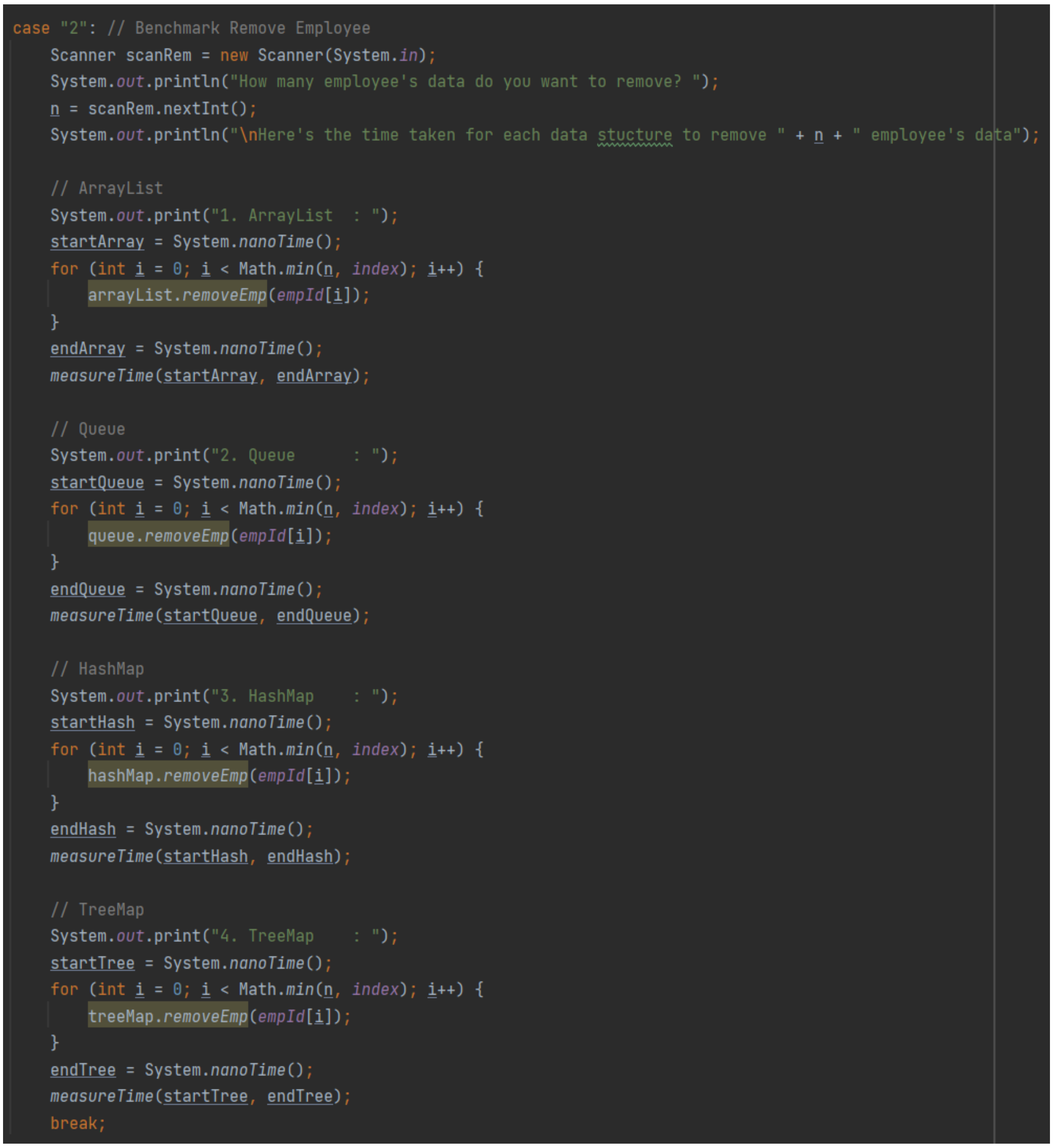
Screenshots of Codes for Methods

Add Employee Method



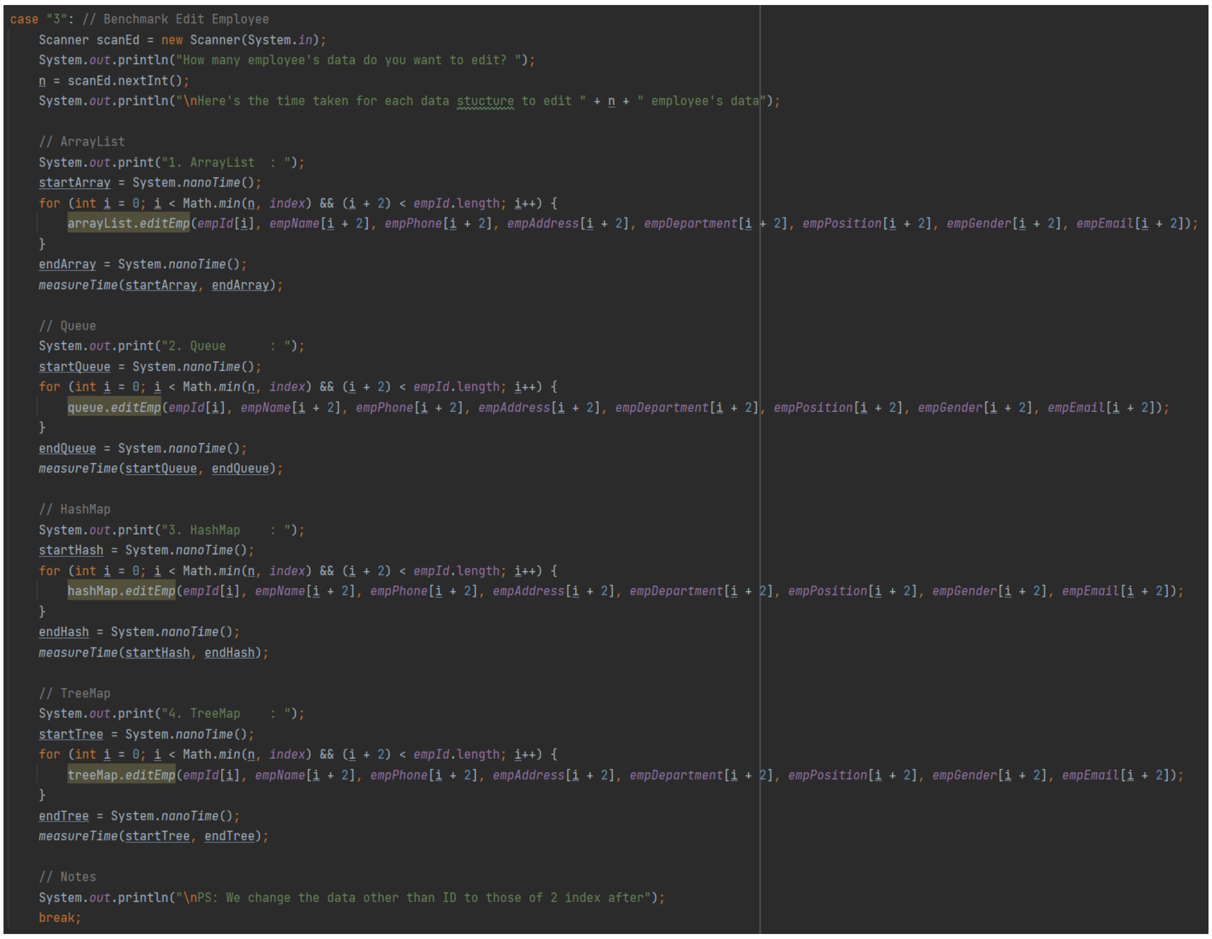
```java
switch (command) {
    case "1": // Benchmark Add Employee
        Scanner scanAdd = new Scanner(System.in);
        System.out.println("How many employee's data do you want to add? ");
        n = scanAdd.nextInt();
        System.out.println("\nHere's the time taken for each data stucture to add " + n + " employee's data");

        // ArrayList
        System.out.print("1. ArrayList  : ");
        startArray = System.nanoTime();
        for (int i = 0; i < Math.min(n, index); i++) {
            arrayList.addEmp(empId[i], empName[i], empPhone[i], empAddress[i], empDepartment[i], empPosition[i], empGender[i], empEmail[i]);
        }
        endArray = System.nanoTime();
        measureTime(startArray, endArray);

        // Queue
        System.out.print("2. Queue      : ");
        startQueue = System.nanoTime();
        for (int i = 0; i < Math.min(n, index); i++) {
            queue.addEmp(empId[i], empName[i], empPhone[i], empAddress[i], empDepartment[i], empPosition[i], empGender[i], empEmail[i]);
        }
        endQueue = System.nanoTime();
        measureTime(startQueue, endQueue);

        // HashMap
        System.out.print("3. HashMap    : ");
        startHash = System.nanoTime();
        for (int i = 0; i < Math.min(n, index); i++) {
            hashMap.addEmp(empId[i], empName[i], empPhone[i], empAddress[i], empDepartment[i], empPosition[i], empGender[i], empEmail[i]);
        }
        endHash = System.nanoTime();
        measureTime(startHash, endHash);

        // TreeMap
        System.out.print("4. TreeMap    : ");
        startTree = System.nanoTime();
        for (int i = 0; i < Math.min(n, index); i++) {
            treeMap.addEmp(empId[i], empName[i], empPhone[i], empAddress[i], empDepartment[i], empPosition[i], empGender[i], empEmail[i]);
        }
        endTree = System.nanoTime();
        measureTime(startTree, endTree);
        break;
```

Remove Employee Method



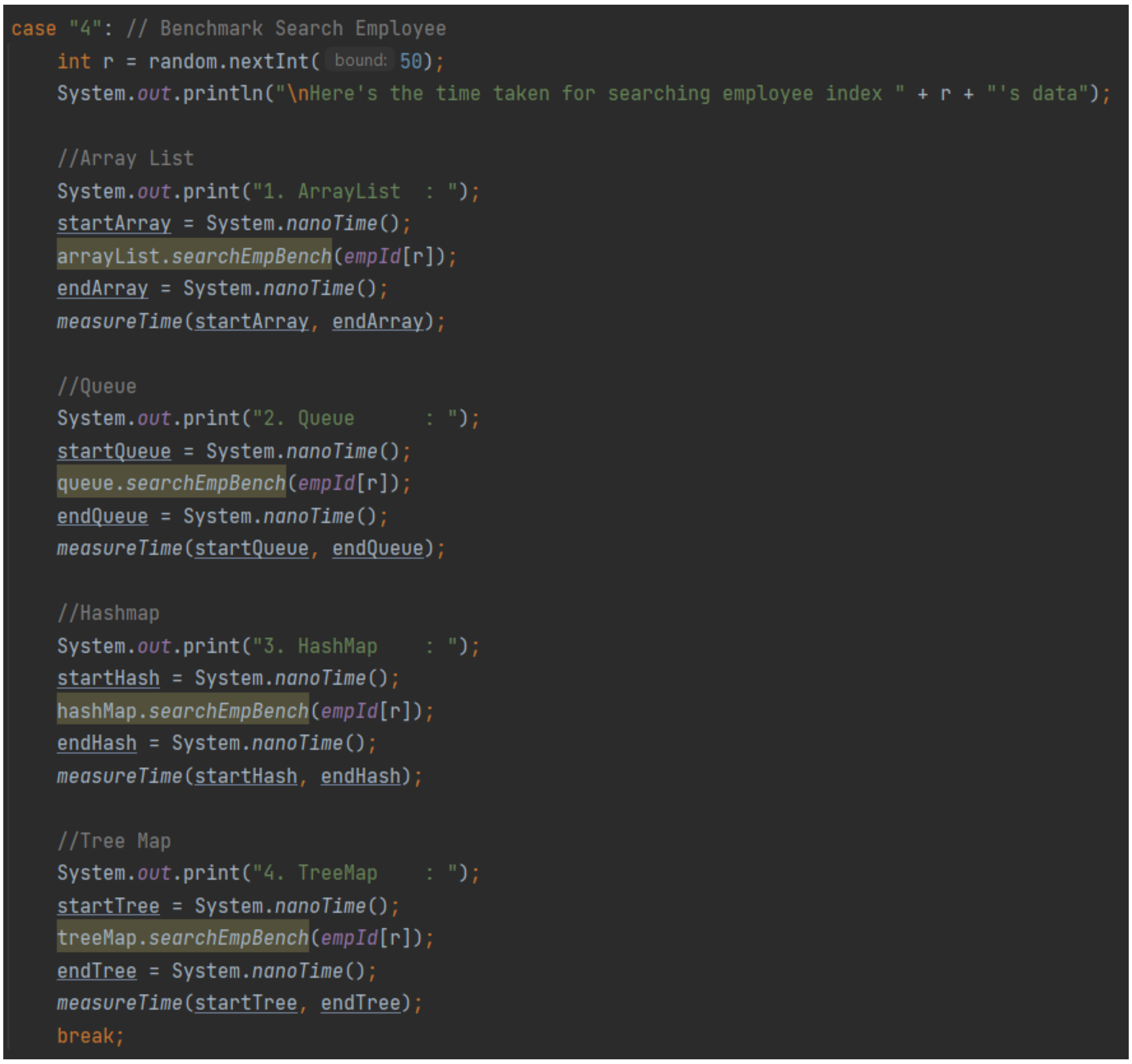
```
case "2": // Benchmark Remove Employee
    Scanner scanRem = new Scanner(System.in);
    System.out.println("How many employee's data do you want to remove? ");
    n = scanRem.nextInt();
    System.out.println("\nHere's the time taken for each data stucture to remove " + n + " employee's data");

    // ArrayList
    System.out.print("1. ArrayList  : ");
    startArray = System.nanoTime();
    for (int i = 0; i < Math.min(n, index); i++) {
        arrayList.removeEmp(empId[i]);
    }
    endArray = System.nanoTime();
    measureTime(startArray, endArray);

    // Queue
    System.out.print("2. Queue      : ");
    startQueue = System.nanoTime();
    for (int i = 0; i < Math.min(n, index); i++) {
        queue.removeEmp(empId[i]);
    }
    endQueue = System.nanoTime();
    measureTime(startQueue, endQueue);

    // HashMap
    System.out.print("3. HashMap    : ");
    startHash = System.nanoTime();
    for (int i = 0; i < Math.min(n, index); i++) {
        hashMap.removeEmp(empId[i]);
    }
    endHash = System.nanoTime();
    measureTime(startHash, endHash);

    // TreeMap
    System.out.print("4. TreeMap    : ");
    startTree = System.nanoTime();
    for (int i = 0; i < Math.min(n, index); i++) {
        treeMap.removeEmp(empId[i]);
    }
    endTree = System.nanoTime();
    measureTime(startTree, endTree);
    break;
```

Edit Employee Method



```
case "3": // Benchmark Edit Employee
    Scanner scanEd = new Scanner(System.in);
    System.out.println("How many employee's data do you want to edit? ");
    n = scanEd.nextInt();
    System.out.println("\nHere's the time taken for each data stucture to edit " + n + " employee's data");

    // ArrayList
    System.out.print("1. ArrayList  : ");
    startArray = System.nanoTime();
    for (int i = 0; i < Math.min(n, index) && (i + 2) < empId.length; i++) {
        arrayList.editEmp(empId[i], empName[i + 2], empPhone[i + 2], empAddress[i + 2], empDepartment[i + 2], empPosition[i + 2], empGender[i + 2], empEmail[i + 2]);
    }
    endArray = System.nanoTime();
    measureTime(startArray, endArray);

    // Queue
    System.out.print("2. Queue      : ");
    startQueue = System.nanoTime();
    for (int i = 0; i < Math.min(n, index) && (i + 2) < empId.length; i++) {
        queue.editEmp(empId[i], empName[i + 2], empPhone[i + 2], empAddress[i + 2], empDepartment[i + 2], empPosition[i + 2], empGender[i + 2], empEmail[i + 2]);
    }
    endQueue = System.nanoTime();
    measureTime(startQueue, endQueue);

    // HashMap
    System.out.print("3. HashMap    : ");
    startHash = System.nanoTime();
    for (int i = 0; i < Math.min(n, index) && (i + 2) < empId.length; i++) {
        hashMap.editEmp(empId[i], empName[i + 2], empPhone[i + 2], empAddress[i + 2], empDepartment[i + 2], empPosition[i + 2], empGender[i + 2], empEmail[i + 2]);
    }
    endHash = System.nanoTime();
    measureTime(startHash, endHash);

    // TreeMap
    System.out.print("4. TreeMap    : ");
    startTree = System.nanoTime();
    for (int i = 0; i < Math.min(n, index) && (i + 2) < empId.length; i++) {
        treeMap.editEmp(empId[i], empName[i + 2], empPhone[i + 2], empAddress[i + 2], empDepartment[i + 2], empPosition[i + 2], empGender[i + 2], empEmail[i + 2]);
    }
    endTree = System.nanoTime();
    measureTime(startTree, endTree);

    // Notes
    System.out.println("\nPS: We change the data other than ID to those of 2 index after");
    break;
```

Search Employee Method



```
case "4": // Benchmark Search Employee
    int r = random.nextInt( bound: 50);
    System.out.println("\nHere's the time taken for searching employee index " + r + "'s data");

    //Array List
    System.out.print("1. ArrayList  : ");
    startArray = System.nanoTime();
    arrayList.searchEmpBench(empId[r]);
    endArray = System.nanoTime();
    measureTime(startArray, endArray);

    //Queue
    System.out.print("2. Queue      : ");
    startQueue = System.nanoTime();
    queue.searchEmpBench(empId[r]);
    endQueue = System.nanoTime();
    measureTime(startQueue, endQueue);

    //Hashmap
    System.out.print("3. HashMap    : ");
    startHash = System.nanoTime();
    hashMap.searchEmpBench(empId[r]);
    endHash = System.nanoTime();
    measureTime(startHash, endHash);

    //Tree Map
    System.out.print("4. TreeMap    : ");
    startTree = System.nanoTime();
    treeMap.searchEmpBench(empId[r]);
    endTree = System.nanoTime();
    measureTime(startTree, endTree);
    break;
```

Analysis of the Time and Space Complexity

- **Time Complexity**

We tested each method with different data structures 6 times, with different indexes, starting from 5, 50, 100 and 250.

- **Add Employee**

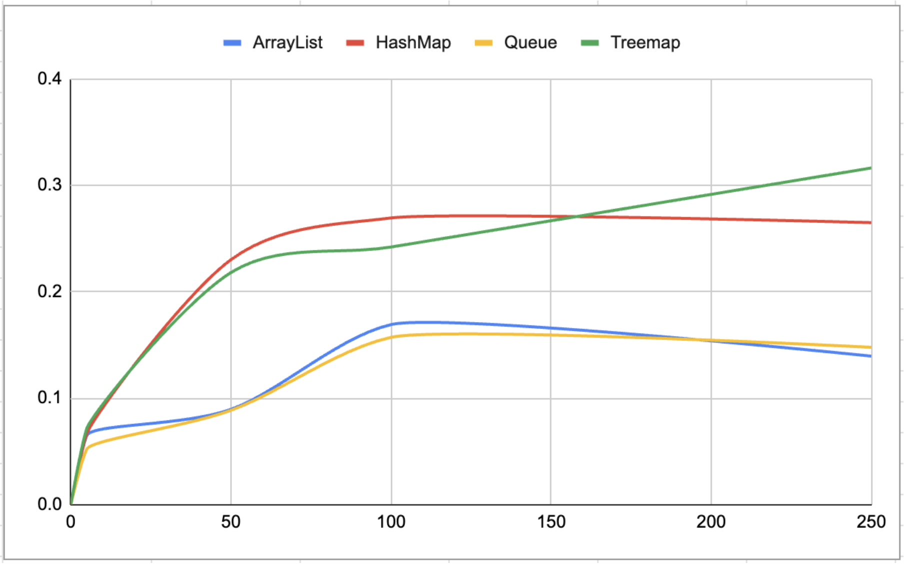
Data Structures	Time Taken to Add 5 Employees (in milliseconds)						Average
ArrayList	0.071292	0.071292	0.071292	0.071292	0.071292	0.032916	0.0649
HashMap	0.080917	0.050667	0.071625	0.069167	0.063875	0.065417	0.0669
Queue	0.043625	0.068208	0.044333	0.042292	0.078333	0.035792	0.0521
TreeMap	0.069833	0.065958	0.072	0.074666	0.070958	0.076125	0.0716

Data Structures	Time Taken to Add 50 Employees (in milliseconds)						Average
ArrayList	0.094	0.072417	0.112667	0.078375	0.094208	0.084625	0.0894
HashMap	0.278833	0.274542	0.238042	0.223042	0.1985	0.167875	0.2301
Queue	0.097042	0.060667	0.100958	0.093459	0.075875	0.10325	0.0885
TreeMap	0.655208	0.115875	0.138	0.136708	0.134666	0.127291	0.218

Data Structures	Time Taken to Add 100 Employees (in milliseconds)						Average
ArrayList	0.099083	0.116125	0.200292	0.181459	0.145042	0.272541	0.1691
HashMap	0.274167	0.321583	0.268042	0.2675	0.254792	0.22975	0.2693
Queue	0.157375	0.162875	0.147375	0.216	0.060583	0.198417	0.1571
TreeMap	0.182458	0.243541	0.2765	0.286041	0.167875	0.2955	0.242

Data Structures	Time Taken to Add 250 Employees (in milliseconds)						Average
ArrayList	0.1303	0.1345	0.1111	0.1944	0.1316	0.1344	0.1394
HashMap	0.1565	0.3234	0.1141	0.3502	0.2692	0.3761	0.2649
Queue	0.0934	0.0885	0.0994	0.2446	0.1581	0.2022	0.1477
TreeMap	0.4101	0.1966	0.1266	0.3495	0.3045	0.5104	0.3163

➔ **The graph**



- Remove Employee

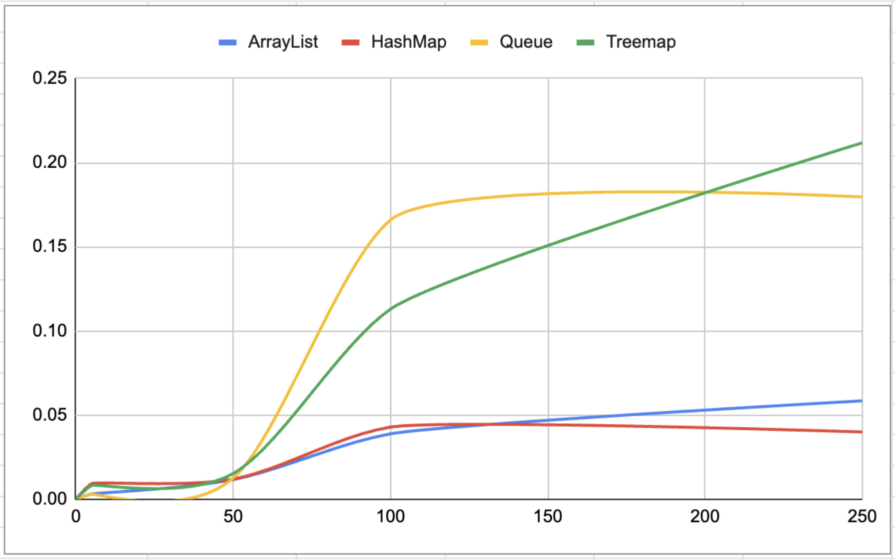
Data Structures	Time Taken to Remove 5 Employees (in milliseconds)						Average
ArrayList	0.0049	0.0027	0.0033	0.0025	0.0029	0.003	0.0032
HashMap	0.014	0.0079	0.0074	0.0071	0.0069	0.0122	0.0093
Queue	0.0047	0.0027	0.0028	0.0026	0.0025	0.0029	0.003
TreeMap	0.0125	0.0073	0.0067	0.0071	0.007	0.0086	0.0082

Data Structures	Time Taken to Remove 50 Employees (in milliseconds)						Average
ArrayList	0.0226	0.0132	0.0093	0.0089	0.0064	0.0098	0.0117
HashMap	0.0144	0.0152	0.0094	0.0092	0.0147	0.0091	0.012
Queue	0.0127	0.0135	0.0125	0.0135	0.0098	0.0136	0.0126
TreeMap	0.0112	0.0255	0.0115	0.0117	0.0129	0.0187	0.0153

Data Structures	Time Taken to Remove 100 Employees (in milliseconds)						Average
ArrayList	0.0341	0.0362	0.0878	0.0268	0.0164	0.0318	0.03885
HashMap	0.0716	0.0625	0.0264	0.0253	0.0452	0.0261	0.04285
Queue	0.1357	0.1309	0.1128	0.2247	0.1514	0.2407	0.1660333333
TreeMap	0.1442	0.0835	0.0904	0.1173	0.0716	0.171	0.113

Data Structures	Time Taken to Remove 250 Employees (in milliseconds)						Average
ArrayList	0.0655	0.0502	0.075	0.0601	0.0624	0.0375	0.0585
HashMap	0.0342	0.0405	0.0617	0.0363	0.0382	0.0291	0.04
Queue	0.1449	0.172	0.2504	0.2364	0.1486	0.1255	0.1796
TreeMap	0.2745	0.1623	0.2451	0.3337	0.1292	0.1259	0.2118

➔ The graph



- Edit Employee

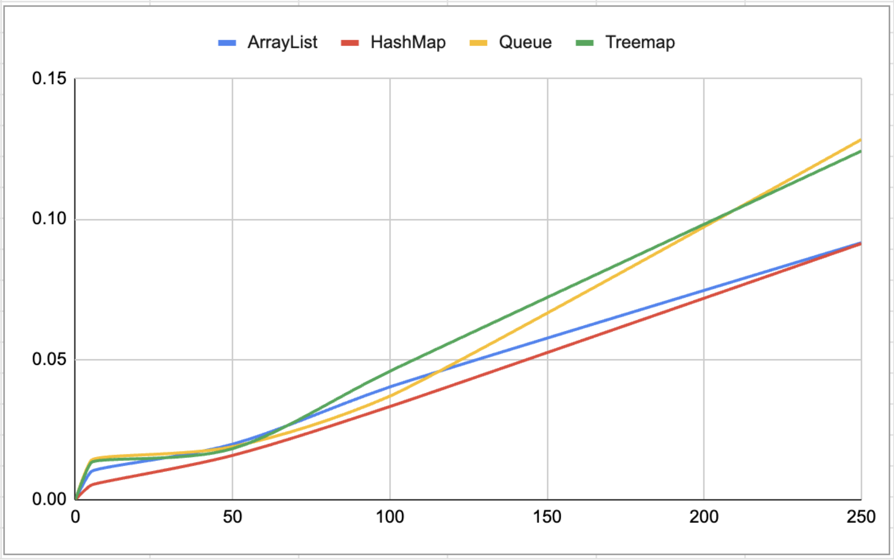
Data Structures	Time Taken to Edit 5 Employees (in milliseconds)						Average
ArrayList	0.01	0.010708	0.009666	0.010042	0.01025	0.009458	0.01
HashMap	0.006416	0.00475	0.005584	0.004292	0.0055	0.004416	0.0052
Queue	0.01425	0.013208	0.015083	0.015	0.014709	0.011709	0.014
TreeMap	0.014666	0.012708	0.014625	0.011375	0.0135	0.0125	0.0132

Data Structures	Time Taken to Edit 50 Employees (in milliseconds)						Average
ArrayList	0.017375	0.016	0.020667	0.029333	0.021375	0.013833	0.0198
HashMap	0.015459	0.011625	0.018792	0.018833	0.019125	0.011042	0.0158
Queue	0.017583	0.013875	0.022	0.022375	0.022042	0.015709	0.0189
TreeMap	0.016334	0.013708	0.022167	0.021458	0.021208	0.014125	0.0182

Data Structures	Time Taken to Edit 100 Employees (in milliseconds)						Average
ArrayList	0.04575	0.052916	0.024416	0.068167	0.026083	0.024166	0.0402
HashMap	0.03525	0.056625	0.019917	0.048042	0.0195	0.01975	0.0332
Queue	0.041459	0.04	0.026209	0.046458	0.039208	0.027834	0.0369
TreeMap	0.0395	0.037875	0.050875	0.042333	0.060459	0.04375	0.0458

Data Structures	Time Taken to Edit 250 Employees (in milliseconds)						Average
ArrayList	0.065042	0.108042	0.093916	0.106375	0.068875	0.107125	0.0916
HashMap	0.075792	0.095958	0.114292	0.086417	0.08875	0.08675	0.0913
Queue	0.192458	0.12225	0.111333	0.105875	0.131166	0.107292	0.1284
TreeMap	0.193791	0.130625	0.095666	0.10675	0.10725	0.112	0.1243

➔ The graph

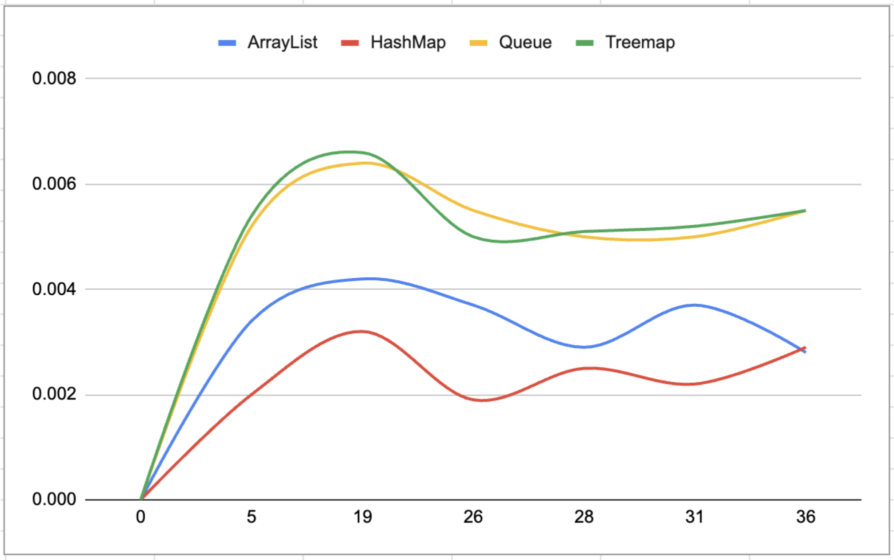


For the search feature, we benchmark it searching for random indexes of data using the java.util.Random.

- **Search Employee**

Data Structures	Time Taken to Search for Random Employee						Average
	Index 19	Index 26	Index 5	Index 31	Index 36	Index 28	
ArrayList	0.004199	0.003701	0.0034	0.003701	0.0028	0.002899	0.0068
HashMap	0.0032	0.0019	0.002001	0.0022	0.0029	0.002499	0.0206
Queue	0.0064	0.0055	0.005201	0.005	0.0055	0.004999	0.0031
TreeMap	0.0066	0.005	0.005399	0.0052	0.0055	0.0051	0.0173

➔ **The graph**



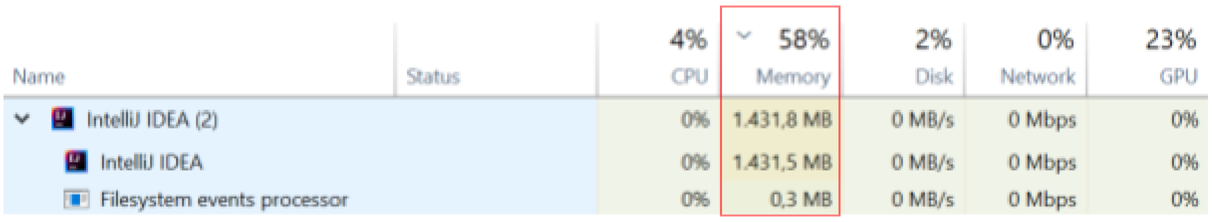
The conclusion of our time complexity is as below:

Method	Num of Data	ArrayList	HashMap	Queue	TreeMap
Add Employee	5	0.0649	0.0669	0.0521	0.0716
	50	0.0894	0.2301	0.0885	0.218
	100	0.1691	0.2693	0.1571	0.242
	250	0.1394	0.2649	0.1477	0.3163
Remove Employee	5	0.0032	0.0093	0.003	0.0082
	50	0.0117	0.012	0.0126	0.0153
	100	0.0389	0.0429	0.166	0.113
	250	0.0585	0.04	0.1796	0.2118
Edit Employee	5	0.01	0.0052	0.014	0.0132
	50	0.0198	0.0158	0.0189	0.0182
	100	0.0402	0.0332	0.0369	0.0458
	250	0.0916	0.0913	0.1284	0.1243
Search Employee	index 5	0.0034	0.002001	0.005201	0.005399
	index 19	0.004199	0.0032	0.0064	0.0066
	index 26	0.003701	0.0019	0.0055	0.005
	index 28	0.002899	0.002499	0.004999	0.0051
	index 31	0.003701	0.0022	0.005	0.0052
	index 36	0.0028	0.0029	0.0055	0.0055

- **Space Complexity**

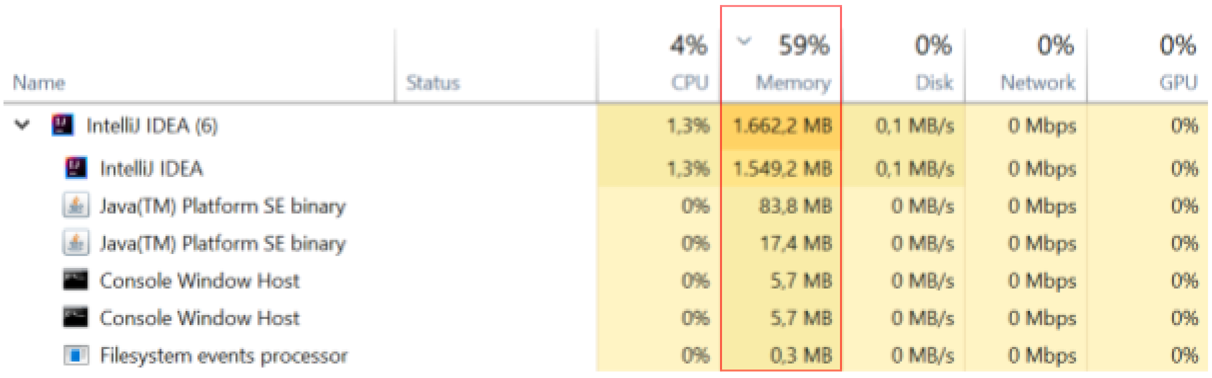
  We check the space complexity by looking through Task Manager and seeing how much space it takes for a method to run. We calculate it by subtracting the memory used after running the method with the memory used before the method is run.

  For example: Before any methods are run, the memory used is:



Name	Status	4% CPU	58% Memory	2% Disk	0% Network	23% GPU
IntelliJ IDEA (2)		0%	1.431,8 MB	0 MB/s	0 Mbps	0%
IntelliJ IDEA		0%	1.431,5 MB	0 MB/s	0 Mbps	0%
Filesystem events processor		0%	0,3 MB	0 MB/s	0 Mbps	0%

  After a method has been run, the memory used can be:



Name	Status	4% CPU	59% Memory	0% Disk	0% Network	0% GPU
IntelliJ IDEA (6)		1,3%	1.662,2 MB	0,1 MB/s	0 Mbps	0%
IntelliJ IDEA		1,3%	1.549,2 MB	0,1 MB/s	0 Mbps	0%
Java(TM) Platform SE binary		0%	83,8 MB	0 MB/s	0 Mbps	0%
Java(TM) Platform SE binary		0%	17,4 MB	0 MB/s	0 Mbps	0%
Console Window Host		0%	5,7 MB	0 MB/s	0 Mbps	0%
Console Window Host		0%	5,7 MB	0 MB/s	0 Mbps	0%
Filesystem events processor		0%	0,3 MB	0 MB/s	0 Mbps	0%

So, based on the two example data, the total memory a method would require is:
1.662,2 MB - 1.431,8 MB = 230,4 MB

➔ **The comparison** (these numbers are an approximate, using megabytes)

Methods	ArrayList	HashMap	Queue	Treemap
Add Employee	108	320	354	126
Remove Employee	212	332	429	190
Edit Employee	125	316	356	120
Search Employee	121	319	424	117

<u>Conclusion</u>

After analysing the time complexity and space complexity of running our program with different data structures, in this case ArrayList, HashMap, Queue, and TreeMap, we came to the conclusion that:

1. Add Employee Method
   - ➔ The fastest data structures for adding employees would be either **Queue** or **ArrayList**. They both have similar time complexity with Queue the fastest in adding small data, but as the data got larger, ArrayList took over as the fastest data structure.
   - ➔ The data structure with the lowest space complexity is ArrayList

   So, the recommended data structure for adding employees after taking time and space complexity into consideration is Arraylist.

2. Remove Employee Method
   - ➔ The recommended data structures for removing employees would be either **HashMap** or **ArrayList**. HashMap has lower time complexity when removing small data, but ArrayList might be a better choice when removing large data.
   - ➔ The data structure with the lowest space complexity is **TreeMap**.

   However, looking at the data overall, it is more recommended to use an Arraylist, as although it isn't the least memory taking data structure, in comparison to HashMap, it is much more efficient.

3. Edit Employee Method
   - ➔ The recommended data structure when it comes to editing is **HashMap**. However, **ArrayList** is also recommended because apparently according to our analysis, when the data got larger, ArrayList and hashmap both have very similar time complexity.
   - ➔ The data structure with the lowest space complexity is **TreeMap**.

   So, the recommended data structure for editing employees' data is ArrayList, as it is the fastest when handling large data and also has similar space complexity to TreeMap.

4. Search Employee Method
   - ➔ The recommended data structure for searching one among many pieces of data, is **HashMap**. It has the lowest time complexity compared to the other data structures.
   - ➔ The data structure with the lowest space complexity is **TreeMap**.

   From the analysis, both maps are efficient in doing searches. So, it depends on what the user needs. If they are looking for the fastest, then HashMap is the answer. But, if they are looking for the least memory consuming, then TreeMap would be the best.

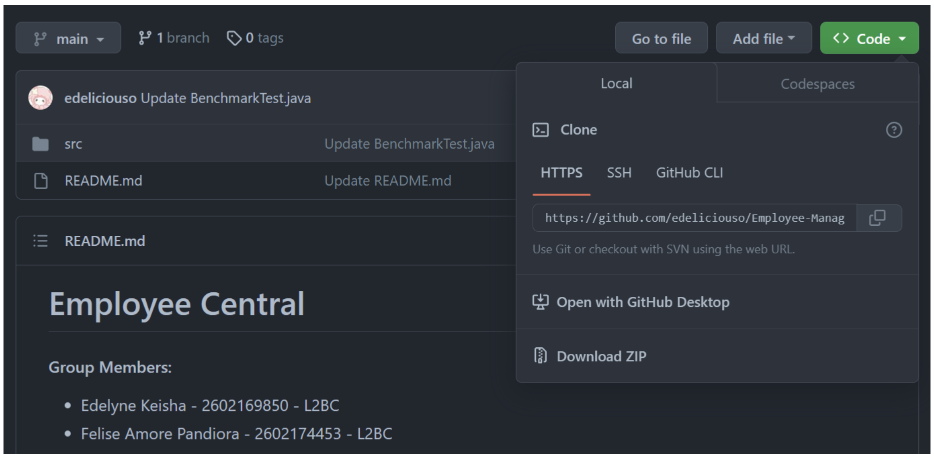
The results of benchmarking may differ from one to another as it also takes your device's hardware, operation system, thermal throttling, power management, and system load into consideration. But the overall conclusion will mostly be the same.

Appendices

Program Manual

1. Open the Github link to the project: https://github.com/edeliciouso/Employee-Management-System

2. Download the repository. To do this, click on the green button in the repo with the writing “Code” and press the Download ZIP button. This ensures that we will get the compressed file of all the files in the repository.



edeliciouso Update BenchmarkTest.java

src	Update BenchmarkTest.java
README.md	Update README.md

README.md

Employee Central

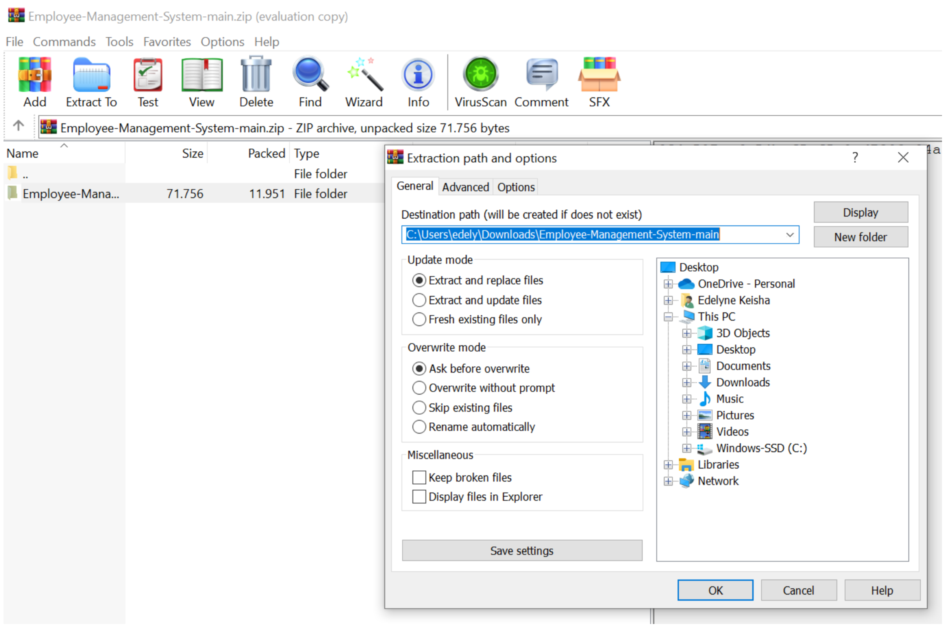
Group Members:

- Edelyne Keisha - 2602169850 - L2BC
- Felise Amore Pandiora - 2602174453 - L2BC

https://github.com/edeliciouso/Employee-Manag

Use Git or checkout with SVN using the web URL.

3. Extract the ZIP file. After downloading the file, simply extract it using an application on your computer and put it in a folder where you can find it.



Employee-Management-System-main.zip (evaluation copy)

Employee-Management-System-main.zip - ZIP archive, unpacked size 71.756 bytes

Name	Size	Packed	Type
..			File folder
Employee-Mana...	71.756	11.951	File folder

Extraction path and options

Destination path (will be created if does not exist)

C:\Users\edely\Downloads\Employee-Management-System-main

Update mode
- [✓ Extract and replace files]
- Extract and update files
- Fresh existing files only

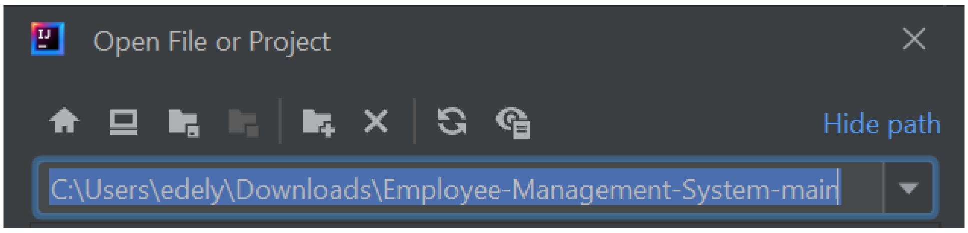
Overwrite mode
- [✓ Ask before overwrite]
- Overwrite without prompt
- Skip existing files
- Rename automatically

Miscellaneous
- Keep broken files
- Display files in Explorer

Desktop
- OneDrive - Personal
- Edelyne Keisha
- This PC
  - 3D Objects
  - Desktop
  - Documents
  - Downloads
  - Music
  - Pictures
  - Videos
  - Windows-SSD (C:)
- Libraries
- Network

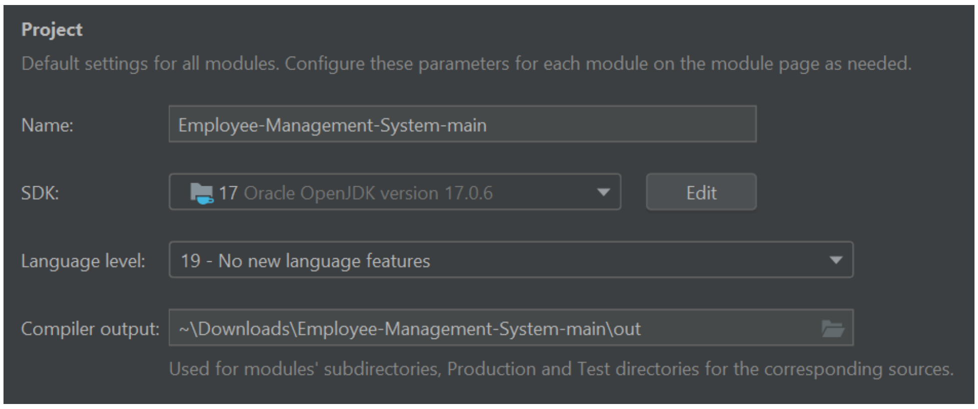
For Windows

4. Open the file. Open the IDE of your choosing (in this case, it's recommended that you use the JetBrains IDE, Intellij IDEA), and locate the file you want to run by going to File > Open and then find the file you want to run.



C:\Users\edely\Downloads\Employee-Management-System-main

5. Set up the environment. Once the file has been imported, you might need to configure some settings or add additional modules and libraries. In Java specifically, it is also important that you have a JDK (Java Development Kit) installed, or else it cannot run.



Project

Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: Employee-Management-System-main

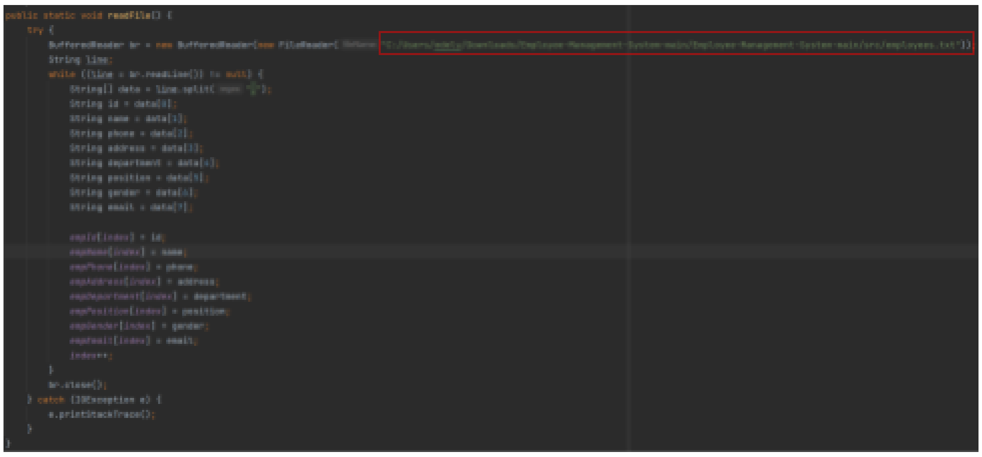
SDK: 17 Oracle OpenJDK version 17.0.6

Language level: 19 - No new language features

Compiler output: ~\Downloads\Employee-Management-System-main\out

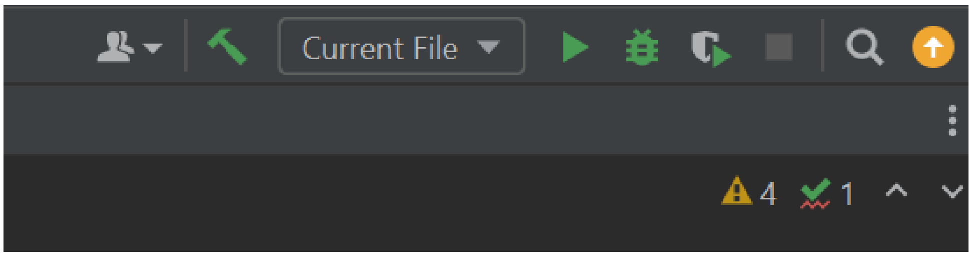
Used for modules' subdirectories, Production and Test directories for the corresponding sources.

In the BenchmarkTest.java file, for the FileReader function, change the fileName based on where you have installed the employee.txt file.



```
public static void readFile() {
    try {
        BufferedReader br = new BufferedReader(new FileReader( fileName: "C:/Users/edely/Downloads/Employee-Management-System-main/Employee-Management-System-main/src/employees.txt"));
        String line;
        while ((line = br.readLine()) != null) {
            String[] data = line.split( regex: ",");
            String id = data[0];
            String name = data[1];
            String phone = data[2];
            String address = data[3];
            String department = data[4];
            String position = data[5];
            String gender = data[6];
            String email = data[7];

            empId[index] = id;
            empName[index] = name;
            empPhone[index] = phone;
            empAddress[index] = address;
            empDepartment[index] = department;
            empPosition[index] = position;
            empGender[index] = gender;
            empEmail[index] = email;
            index++;
        }
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

6. After everything is finished, run the Java file by pressing the green play button on the top right (assuming this is run on Intellij).



Link to GIT Website

https://github.com/edeliciouso/Employee-Management-System

Link to Presentation Files

https://www.canva.com/design/DAFlnvYmgsE/zLLqI0eKeMTNkyj6_F1r-w/edit?utm_content=DAFlnvYmgsE&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton