

EMPLOYEE OPERATION APPLICATION

This project is a employee management system that enables users to import and manage employee information in a company. The system allows the user to import a dataset of employee data from a .csv file, reorganise it into manageable collections, perform calculations (i.e. average salary), filter and display the information. It consists of the Employee class, EmployeeOperations class and the EmployeeToInfo function interface.

Employee class:

The Employee class is a simple representation of an employee in a company. It contains four private attributes: name, age, department, and salary. These attributes are used to store the employee's personal information.

The class includes a constructor that takes four parameters: name, age, department, and salary. It initializes the corresponding attributes when an instance of the class is created.

Additionally, the class provides four getter methods:

- getName(): Returns the employee's name.
- getAge(): Returns the employee's age.
- getDepartment(): Returns the employee's department.
- getSalary(): Returns the employee's salary.

These getter methods allow accessing the employee's information, while the attributes remain private, maintaining encapsulation.

Employee.java × EmployeeOperations.java EmployeeToInfo.java

```
1 public class Employee {
2     String name;
3     int age;
4     String department;
5     double salary;
6
7     public Employee(String name, int age, String department, double salary) {
8         this.name = name;
9         this.age = age;
10        this.department = department;
11        this.salary = salary;
12    }
13
14    public String getName() {
15        return name;
16    }
17
18    public int getAge() {
19        return age;
20    }
21
22    public String getDepartment() {
23        return department;
24    }
25
26    public double getSalary() {
27        return salary;
28    }
29 }
```

EmployeeOperations class:

The EmployeeOperations class contains a set of methods that operate on a dataset of Employee instances. It is responsible for managing and manipulating the list of employees and performing various operations on the data.

- `main()` method: This method demonstrates the usage of the other methods in the class. It retrieves the employee information, calculates the average salary, and prints out the employees above a certain age threshold.
- `importEmployeeDataset()` method: Reads employee data from a CSV file named "Employees.csv" and imports it into a list of Employee objects. Each row in the CSV file represents an employee with their name, age, department, and salary.
- `getEmployeeList()` method: Returns the list of employees stored in the employees static variable.
- `getEmployeeInfoList()` method: Iterates over the list of employees and uses the `EmployeeToInfo` function to create a list of strings containing the employee information (name and department).
- `getAverageSalary()` method: Calculates the average salary of all employees using Java's Stream API.
- `getEmployeesAboveAge()` method: Takes an age threshold as a parameter and returns a list of employees with an age greater than the threshold, using Java's Stream API for filtering.

This class provides an encapsulation of operations for handling and processing employee data within a program.

```

1  import java.util.*;
2  import java.io.BufferedReader;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  public class EmployeeOperations {
7      static List<Employee> employees = importEmployeeDataset("Employees.csv");
8      public static void main(String[] args) {
9          List<String> employeeInfoList = getEmployeeInfoList();
10
11         employeeInfoList.forEach(System.out::println);
12
13         double averageSalary = getAverageSalary();
14
15         System.out.printf("\nAverage Salary: $%.2f\n", averageSalary);
16
17         int ageThreshold = 30;
18         List<Employee> employeesAbove30 = getEmployeesAboveAge(30);
19         System.out.println("\nEmployees above 30:");
20         employeesAbove30.forEach(employee -> System.out.println(employee.getName()));
21     }
22
23     @ private static List<Employee> importEmployeeDataset(String fileName) {
24         String line;
25         String[] row;
26         List<Employee> employees = new ArrayList<>();
27
28         try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
29             while ((line = br.readLine()) != null) {
30                 row = line.split(",");
31                 employees.add(new Employee(row[0], Integer.parseInt(row[1]), row[2], Double.parseDouble(row[3])));
32             }
33         } catch (IOException e) {
34             e.printStackTrace();
35         }
36
37         return employees;
38     }
39
40     public static List<Employee> getEmployeeList() { return employees; }
41
42     private static List<String> getEmployeeInfoList() {
43         EmployeeToInfo employeeToInfo = new EmployeeToInfo();
44         List<String> employeeInfoList = new ArrayList<>();
45
46         for (Employee employee : employees) {
47             String employeeInfo = employeeToInfo.apply(employee);
48             employeeInfoList.add(employeeInfo);
49         }
50
51         return employeeInfoList;
52     }
53
54     private static Double getAverageSalary() {
55         return(employees.stream()
56             .mapToDouble(Employee::getSalary)
57             .average()
58             .getAsDouble());
59     }
60
61     private static List<Employee> getEmployeesAboveAge(int ageThreshold) {
62         return(employees.stream()
63             .filter(e -> e.getAge() > ageThreshold)
64             .toList());
65     }
66 }

```


EmployeeToInfo function interface:

The EmployeeToInfo class implements the Function interface, which is a functional interface in Java that takes an input of type T and returns an output of type R. In this case, EmployeeToInfo is a function that takes an Employee object as input and returns a String as output.

The apply method in EmployeeToInfo is the implementation of the Function interface's abstract method. It does the following:

- Obtains the list of employees from the EmployeeOperations class.
- Iterates over the list of employees to find the employee with the same name as the target employee provided as input.
- If an employee is found, it returns a concatenated string containing the employee's name and department.
- If no matching employee is found, it returns null.

The purpose of this function is to retrieve an employee's information (name and department) based on the input employee name and present it as a formatted string.



```
1 import java.util.List;
2 import java.util.function.Function;
3
4 public class EmployeeToInfo implements Function<Employee, String> {
5     @Override
6     public String apply(Employee targetEmployee) {
7         List<Employee> employees = EmployeeOperations.getEmployeeList();
8         for (Employee employee : employees) {
9             if (employee.getName().equals(targetEmployee.getName())) {
10                 return employee.getName() + " (" + employee.getDepartment() + ")";
11             }
12         }
13         return null;
14     }
15 }
```

The program demonstrates the usage of the Function interface in Java, which represents a function that takes an input of type T and returns an output of type R. The Function interface is a functional interface that contains only one abstract method, `apply(T t)`. By implementing this interface, you can define a custom behavior for the function.

In this program, a Function is implemented as a separate class called `EmployeeToInfo`. This class takes an `Employee` object as input and produces a `String` containing the employee's name and department as output. The `apply` method is used to perform this transformation.

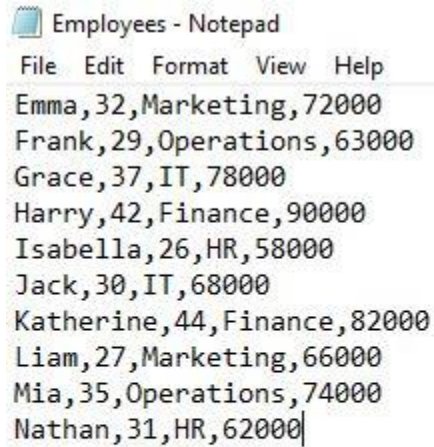
The Function interface offers the following characteristics and benefits:

- **Reusability:** By creating a Function instance, you can reuse the same behavior for different inputs without needing to repeat the code.
- **Flexibility:** The Function interface can be implemented in multiple ways, such as lambda expressions, method references, or anonymous inner classes.
- **Functional Programming Support:** The Function interface encourages a functional programming style in Java, allowing you to treat functions as first-class citizens, enabling immutability, and facilitating methods like `map`, `filter`, and `reduce`.

To use the Function interface, you need to define a lambda expression, method reference, or anonymous inner class that implements the `apply` method. In this program, the `apply` method takes an `Employee` as input, iterates through a list of employees to find a match, and returns a `String` containing the employee's name and department.

Overall, the Function interface in Java enables a more concise and declarative programming style, allowing developers to define reusable and flexible functions that can be easily applied to various contexts and inputs.

Employee Dataset CSV:

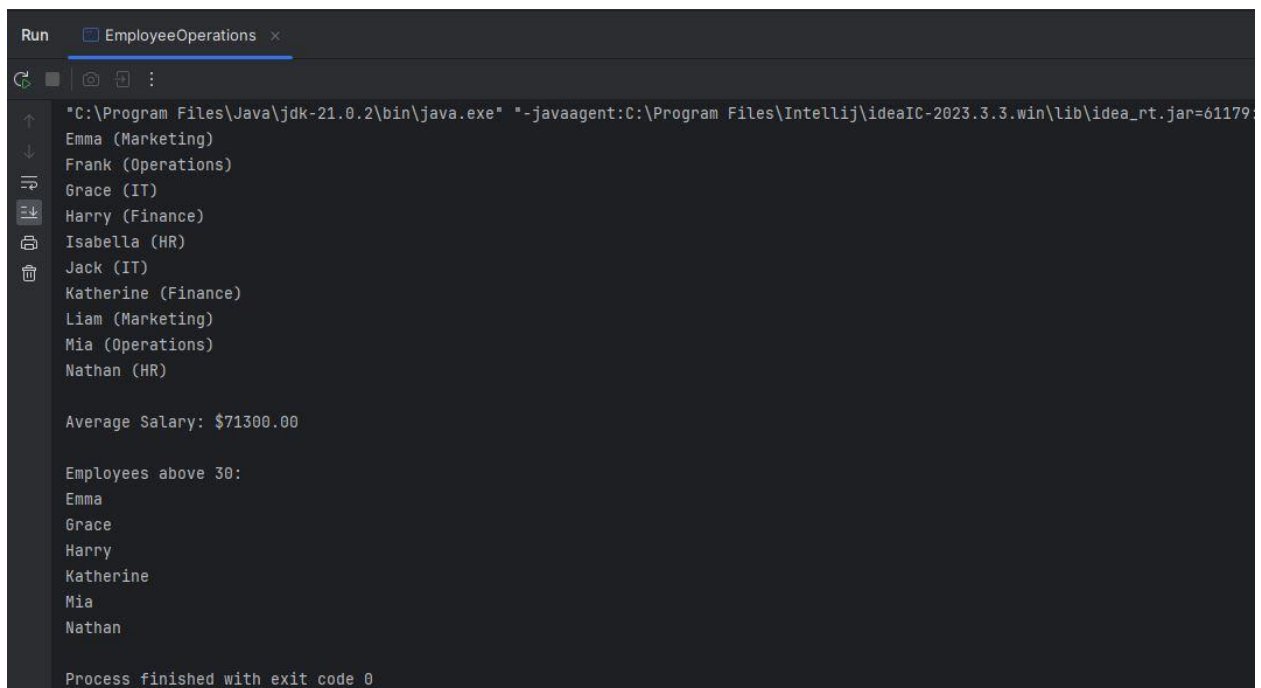


Employees - Notepad

File Edit Format View Help

Emma,32,Marketing,72000
Frank,29,Operations,63000
Grace,37,IT,78000
Harry,42,Finance,90000
Isabella,26,HR,58000
Jack,30,IT,68000
Katherine,44,Finance,82000
Liam,27,Marketing,66000
Mia,35,Operations,74000
Nathan,31,HR,62000

Output:



Run EmployeeOperations x

"C:\Program Files\Java\jdk-21.0.2\bin\java.exe" "-javaagent:C:\Program Files\IntelliJ\ideaIC-2023.3.3.win\lib\idea_rt.jar=61179:..."

Emma (Marketing)
Frank (Operations)
Grace (IT)
Harry (Finance)
Isabella (HR)
Jack (IT)
Katherine (Finance)
Liam (Marketing)
Mia (Operations)
Nathan (HR)

Average Salary: \$71300.00

Employees above 30:
Emma
Grace
Harry
Katherine
Mia
Nathan

Process finished with exit code 0