**DESCRIPTION**

**PROJECT:**

Create a class Employee with the following member name: empId, salary, hireDate, jobPosition, contactNumber, address. Observe here that Date and Address are also the classes. The Date and Address are also the classes with appropriate constructors to set the fields of Employee.

Create a Test class with a main () method. Create an employee database of a company having 500 employees. The following method headers are follows:

1.public static void arrangeEmployeeBySalary (Employee e [])

2.public static void getEmployeesByJobPosition (Employee e[], String jp)

3.public static void getEmployeesByHireDate (Employee e[], Date d1, Date d2)

4. public static int foreignEmployeeCount (Employee e [])

5.public static void getEmployeesBySalary(Employee e[], double s1, double s2)

Moreover, you can add appropriate setter/getter methods if required.

**About Employee Database Program:**

In the Employee Program we will create employee database of a company having 500 employees with using an array of objects. The code is a Java program that defines three classes: ‘Employee’, ‘Date’, and ‘Address’. It also contains a ‘Test’ class that creates an array of Employee objects, initializes them with user input, and then performs some operations on the array. In Test Class in which methods of the functions are there. The methods arrange the employees by salary in descending order, sort the employee by manager, sort the employees by hire date, count the foreign employees, and details of the employee by salary. Thus, the program gives the output and gives the details of the employees in this way. Overall, the code demonstrates basic object-oriented programming concepts such as classes, objects, constructors, encapsulation, and inheritance. It also uses Java libraries such as Scanner and Arrays.

**Basic working of the code:**

The code defines three classes: Employee, Date, and Address. The Employee class has various attributes such as name, employee ID, salary, hire date, job position, contact number, and address. The Date class represents a date with day, month, and year attributes, as well as a method to check if it is between a given start and end date. The Address class contains attributes such as street, city, state, country, and zip code. The Test class contains the main method, where the code initializes an array of Employee objects and asks the user to input details for each employee. It creates an employee object for each set of inputs and stores it in the array. After initializing all Employee objects, the code calls several methods to perform various operations on the employees. The methods include arranging employees by salary in descending order, getting employees by job position, counting foreign employees, getting employees by salary range, and getting employees hired between a given start and end date. Each of these methods processes the employee array and prints out the required information.

The code then terminates.

**Details Of Employee Class:**

The ‘Employee’ class has private fields to store employee details such as name, employee ID, salary, hire date, job position, contact number, and address. It has a constructor to initialize these fields and getters to access the fields.

**Details Of Date Class:**

The ‘Date’ class has private fields to store the day, month, and year of a date. It has a constructor to initialize these fields and getters and setters to access and modify the fields. It also has a method to check whether a date is between two other dates and a method to convert the date to a string.

**Details Of Address Class:**

The ‘Address’ class has private fields to store address details such as street, city, state, country, and zip code. It has a constructor to initialize these fields and getters to access the fields.

**Details Of Test Class:**

The ‘Test’ class has a ‘main’ method that creates an array of Employee objects and initializes them with user input. It then calls several methods to perform operations on the array. These methods include sorting the array in descending order by salary, retrieving employees by job position, counting the number of foreign employees, retrieving employees by salary range, and retrieving employees hired between two dates.

**Working of arrangeEmployeeBySalary():**

The method ‘arrangeEmployeeBySalary’ takes an array of ‘Employee’ objects as an input parameter and sorts the elements of the array based on their salaries in ascending order. Here's how the code works:

1. The method begins by checking if the input array ‘e’ is null or has a length of 0. If either condition is true, the method returns without doing anything.
2. If the input array is not null and has at least one element, the method uses the ‘Arrays.sort()’ method to sort the array in ascending order based on the salaries of the employees. The ‘Arrays.sort()’ method uses the natural ordering of the objects in the array, which in this case is defined by the ‘compareTo()’ method of the ‘Employee’ class.
3. Once the array has been sorted, the method prints out the sorted array to the console using a ‘for’ loop that iterates over each element in the array.
4. The method ends and returns control to the calling code. The input array ‘e’ is now sorted in ascending order based on the salaries of the employees.

**Working of getEmployeesByJobPosition();**

The ‘getEmployeesByJobPosition’ method takes an array of ‘Employee’ objects and a ‘String’ job position as its input parameters. It iterates over the array of employees and checks the job position of each employee against the provided job position. If the job position matches, the employee is added to a new array of employees with that same job position. Once all the employees have been checked, the method returns the new array of employees with matching job positions. Here’s a step-by-step breakdown of how the method works:

1. Create an empty array of ‘Employee’ objects called ‘result’.
2. Iterate over the array of ‘Employee’ objects provided as input.
3. For each ‘Employee’ object, check if their job position matches the input ‘String’ job position.
4. If the job position matches, add the ‘Employee’ object to the ‘result’ array.
5. Once all Employee objects have been checked, return the ‘result’ array.

**Working of getEmployeesByHireDate():**

The getEmployeesByHireDate method takes in an array of ‘Employee’ objects ‘e’, and two ‘Date’ objects ‘d1’ and ‘d2’, representing the start and end dates for which the employee hire dates are to be compared. The method first creates an empty ‘Array List’ to store the ‘Employee’ objects whose hire dates fall within the specified range. It then iterates through each ‘Employee’ object in the array ‘e’ and checks whether their hire date is between ‘d1’ and ‘d2’. If the hire date is within this range, the ‘Employee’ object is added to the ‘Array List’. Finally, the method converts the ‘Array List’ to an array of ‘Employee’ objects and prints out the details of each employee in the array.

**Working of foreignEmployeeCount():**

The method ‘getEmployeesBySalary(Employee e[], double s1, double s2)’ takes an array of ‘Employee’ objects ‘e’, and two double values ‘s1’ and ‘s2’. It then loops through each Employee object in the array and checks if their salary falls between the two given values ‘s1 and s2’. If an ‘Employee’ object's salary falls within the specified range, the method adds that ‘Employee’ object to a new array ‘selected Employees’. Once the loop completes, the method prints the details of all the ‘Employee’ objects in the ‘selected Employees’ array, including their name, job position, hire date, and salary. If no ‘Employee’ objects are found within the specified salary range, the method prints a message stating that no employees were found. Here is the step-by-step process:

1. Initialize an empty ‘Employee’ array called ‘selected Employees’.
2. Loop through each ‘Employee’ object in the ‘e’ array.
3. For each ‘Employee’ object, check if their salary falls within the range ‘s1’ and ‘s2’.
4. If an ‘Employee’ object's salary falls within the specified range, add that Employee object to the ‘selected Employees’ array.
5. Once the loop completes, check if the ‘selected Employees’ array is empty.
6. If the ‘selected Employees’ array is not empty, print the details of all the ‘Employee’ objects in the ‘selected Employees’ array, including their name, job position, hire date, and salary.
7. If the ‘selected Employees’ array is empty, print a message stating that no employees were found within the specified salary range.

**Working of main() method:**

The ‘main()’ method is the entry point of the program, and it is where the program starts its execution. In this program, the ‘main()’ method performs the following steps:

1. It creates an array of ‘Employee’ objects with a size of 500.
2. It initializes the ‘Employee’ objects with sample data such as name, hire date, job position, salary, and country.
3. It calls various static methods of the ‘Employee’ class to perform different operations such as sorting employees by salary, filtering employees by job position, filtering employees by hire date range, counting the number of foreign employees, and filtering employees by salary range.
4. It prints the results of these operations to the console using ‘System.out.println()’ statements.

In summary, the ‘main()’ method sets up the initial data, performs various operations on the data, and outputs the results to the console.

**PROGRAM OUTPUT:**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

BY Ishan Nasker

Reg no-2241014043

Section-2241030

Branch-CSIT

-------------------------x----------------------------