# **DATA STRUCTURE AND ALGORITHM (DSA)**

**MINOR PROJECT-1 (2023)**

# EMPLOYEE DATABASE USING CLASS AND OBJECT

# NAME: Ashreeta Patra

# REG. NO.: 2241016258

# SECTION: 2241033

# BRANCH: CSE

# SEM: 2nd

# DATE: 14.05.2023

# PROBLEM:

# To create an Employee Database using class and object.

# Description of my project:

# **Employee Class:**

# The provided Java code defines a class called *Employee* within the package *Ashreeta\_2241016258*. The *Employe* class represents an employee entity and contains several private fields, along with their respective getter and setter methods.

# The fields of the *Employee* class are as follows:

# 1. name (type: String): Represents the name of the employee.

# *2. empId (type: int):* Represents the unique identification number of the employee.

# *3. salary (type: double):* Represents the salary of the employee.

# *4. hireDate` (type: Date):* Represents the date when the employee was hired.

# *5. jobPosition (type: String****):*** Represents the job position or designation of the employee.

# *6. contactNumber (type: String):* Represents the contact number of the employee.

# *7. address (type: Address):* Represents the address of the employee. The `Address` class is assumed to be defined elsewhere.

# The *Employee* class has a constructor that takes arguments for all the fields, allowing the creation of an *Employee* object with initial values for each field.

# Additionally, the *Employee* class provides getter and setter methods for each field, allowing other parts of the code to retrieve and modify the values of the employee's attributes.

# The purpose of this code is to provide a blueprint for creating and manipulating employee objects in a Java program. By encapsulating the employee's attributes within a class, it allows for easier organization and management of employee data. Other parts of the program can create instances of the *Employee* class, set values for its fields, retrieve information about employees, and update their attributes as needed. The code promotes encapsulation and follows the principles of object-oriented programming.

# **Date Class:**

# The provided Java code defines a class called *Date* within the package *Ashreeta\_2241016258*. The *Date* class represents a date and contains private fields to store the day, month, and year values. It also provides getter and setter methods for these fields.

# The fields of the `Date` class are as follows:

# *1. day (type: int):* Represents the day of the date.

# *2. month (type: int):* Represents the month of the date.

# *3. year (type: int):* Represents the year of the date.

# The *Date* class has a constructor that takes arguments for the day, month, and year, allowing the creation of a *Date* object with specific values for each field.

# Additionally, the *Date* class provides getter and setter methods for each field, allowing other parts of the code to retrieve and modify the values of the date attributes.

# The *Date* class also includes a getTime() method that calculates and returns the time in milliseconds since the epoch (January 1, 1970) corresponding to the date stored in the *Date* object. It uses the Calendar class from the *java.util* package to perform this calculation. The method creates a *Calendar* instance, sets the day, month, and year components of the calendar, and then retrieves the time in milliseconds using the getTimeInMillis() method of the *Calendar* class.

# The purpose of this code is to provide a convenient representation of a date and enable operations related to dates, such as retrieving the time in milliseconds. The *Date* class can be used in various applications that require date manipulation, such as calculating durations, performing date comparisons, or managing events based on specific dates.

# **Address Class:**

# The provided Java code defines a class called *Addres*s within the package *Ashreeta\_2241016258*. The *Address* class represents a physical address and contains private fields to store the address details, along with their respective getter and setter methods.

# The fields of the *Address* class are as follows:

# *1. street (type: String*): Represents the street or house number of the address.

# *2. city (type: String):* Represents the city where the address is located.

# *3. state (type: String):* Represents the state or province where the address is located.

# *4. zipCode (type: String):* Represents the zip code or postal code of the address

# The *Address* class has a constructor that takes arguments for all the fields, allowing the creation of an `Address` object with initial values for each field.

# Additionally, the *Address class* provides getter and setter methods for each field, allowing other parts of the code to retrieve and modify the values of the address attributes.

# The purpose of this code is to provide a blueprint for creating and manipulating address objects in a Java program. By encapsulating the address details within a class, it allows for easier organization and management of address data. Other parts of the program can create instances of the *Address* class, set values for its fields, retrieve address information, and update the address attributes as needed. The code promotes encapsulation and follows the principles of object-oriented programming.

# **Test Class:**

# The provided Java code defines a class called *Test* within the package *Ashreeta\_2241016258*. The *Test* class contains a *main method* where various operations related to an employee database are performed.

# Here's a breakdown of what the code does:

# **1. Creating the Employee Database:**

# - The *createEmployeeDatabase* method is called to generate an array of *Employee* objects containing employee details. The array can hold up to 500 employees.

# - The *createEmployee* method is used to create and populate each `Employee` object with unique values based on the provided index.

# **2**. **Arranging Employees by Salary:**

# - The *arrangeEmployeeBySalar*` method is called to arrange the employee details in descending order based on their salary.

# - The sorted employee details are displayed, showing the name and salary of each employee.

# **3. Retrieving Employees by Job Position:**

# - The *getEmployeesByJobPosition* method is called to retrieve and display the details of employees whose job position matches the provided jobPosition parameter (in this case, "manager").

# - The name and job position of each matching employee are printed.

# 4. **Retrieving Employees by Hire Date Range:**

# - The *getEmployeesByHireDate* method is called to retrieve and display the details of employees whose hire date falls within the specified date range.

# - The start date (April 1, 2022) and end date (March 31, 2023) are used to filter the employees.

# - The name and hire date of each matching employee are printed.

# **5. Counting Foreign Employees:**

# - The *foreignEmployeeCount* method is called to count the number of foreign employees in the database.

# - The phone number of each employee is checked, and if it starts with the "+" symbol, it is considered a foreign employee.

# - The count of foreign employees is printed.

# **6. Retrieving Employees by Salary Range:**

# - The *getEmployeesBySalary* method is called to retrieve and display the details of employees whose salary falls within the specified range (150000 INR to 300000 INR).

# - The name and salary of each employee within the given range are printed.

# **7. Date Range Comparison:**

# - The *isDateInRange* method is used to compare dates and determine if a given date falls within a specified date range.

# **8. Random Hire Date Generation:**

# - The *generateRandomHireDate* method is used to generate a random hire date within a specific range (in this case, within the past 5 years).

# - The start and end dates are calculated, and a random hire date is generated between them.

# The purpose of this code is to demonstrate various operations on an employee database, such as sorting employees by salary, filtering employees by job position and hire date, counting foreign employees, and retrieving employees within a specific salary range. It also includes utility methods for date range comparison and random hire date generation. This code provides a foundation for managing and analyzing employee data in a flexible and organized manner.

# OUTPUT:

# Filling the database manually (500 entries) was not practical so generated random employee list using a loop.

# The output of this project is in the order as per the question given by the authority (Whole output isn’t attached cause of huge length).











