# Data Structures Exam: Enterprise

## Description

You need to implement a structure that will be used by enterprise organizations. You are given an Employee class, which has the following properties:

* Guid Id – unique id for each employee. In Java, it's called UUID
* DateTime HireDate – date that we will use for our queries later
* Position Position – enumeration of work positions for the employee
* string FirstName – first name of the employee
* string LastName – last name of the employee
* double Salary – the salary of the employee

You need to support the following operations (and they should be **fast**):

* Add() – Add an employee to the organization. You will need to implement the Contains() methods as well.
* Contains(Employee) – checks if an employee is present in the organization
* Contains(Guid) – checks if an employee with the given Guid exists in the organization
* Count – returns the number of employees in the organization
* Change(Guid, Employee) – find the employee with the given Guid and modify its data. Returns true if the operation is successful
* Fire(Guid) – remove the employee from the organization. Returns true if the operation is successful
* RaiseSalary(int, int) – raise the salary of all employees which are working in the organization for at least that many months, with the given percentage. Returns true if the operation modified at least one employee
* GetByGuid(Guid) – return the employee with the given Guid. If such employee doesn't exist, throw ArgumentException.
* PositionByGuid(Guid) – return the employee position with the given Guid. If such employee doesn't exist, throw InvalidOperationException.
* GetByPosition(Position) – returns all employees with the given position
* GetBySalary(double) – returns all employees with salary higher or equal to the given salary. Throw InvalidOperationException if there are no employees with matching the search.
* GetBySalaryAndPosition(double, Position) – returns all employees with the given salary and position. Throws InvalidOperationException if there are no employees with the given data.
* SearchBySalary(double, double) – search for all employees with a salary in a range. If there are no such employees, return empty collection
* SearchByPosition(IEnumerable<Position>) – returns all employees with the given positions or empty collection. Returns empty collection if there are no such elements
* SearchByFirstName(string) – returns all employees with the given first name
* SearchByNameAndPosition(string, string, Position) – returns all employees with the given first and last names, and with the given position. Returns empty collection if there are no such elements
* AllWithPositionAndMinSalary(Position, double) – returns all employees with the given position and minimal salary. Returns empty collection if there are no such elements

The equivalent exception in java is IllegalArgumentException.

## Input / Output

You are given a **Visual Studio C# project skeleton** (unfinished project) / **IntelliJ Java project** holding the interface IEnterprise, the unfinished classes Enterprise and Person. **Tests** covering the Enterprise **functionality** and **performance**.

Your task is to **finish this class** to make the tests run correctly.

* You are **not allowed to change the tests**.
* You are **not allowed to change the interface**.
* You can add to the Employee class, but don't remove anything.
* You can edit the Enterprise class if it implements the IEnterprise interface.

## Interface

The interface IEnterprise in C# looks like the code below:

|  |
| --- |
| public interface IEnterprise : IEnumerable<Employee>  {  void Add(Employee employee);  bool Change(Guid guid, Employee employee);  bool Fire(Guid guid);  bool RaiseSalary(int months, int percent);  int Count { get; set; }  Employee GetByGuid(Guid guid);  Position PositionByGuid(Guid guid);  IEnumerable<Employee> GetByPosition(Position position);  IEnumerable<Employee> GetBySalary(double minSalary);  IEnumerable<Employee> GetBySalaryAndPosition(double salary, Position position);  IEnumerable<Employee> SearchBySalary(double minSalary, double maxSalary);  IEnumerable<Employee> SearchByPosition(IEnumerable<Position> positions);  IEnumerable<Employee> AllWithPositionAndMinSalary(Position position, double minSalary);  IEnumerable<Employee> SearchByFirstName(string firstName);  IEnumerable<Employee> SearchByNameAndPosition(string firstName, string lastName, Position position);  } |

The interface Enterprise in Java looks like the code below:

|  |
| --- |
| **public interface** IEnterprise **extends** Iterable<Employee> {   **void** add(Employee employee);  **boolean** contains(UUID id);  **boolean** contains(Employee employee);  **boolean** change(UUID id, Employee employee);  **boolean** fire(UUID id);  **boolean** raiseSalary(**int** months, **int** percent);  **int** getCount();   Employee getByUUID(UUID id);  Position positionByUUID(UUID id);  Iterable<Employee> getByPosition(Position position);  Iterable<Employee> getBySalary(**double** minSalary);  Iterable<Employee> getBySalaryAndPosition(**double** salary, Position position);  Iterable<Employee> searchBySalary(**double** minSalary, **double** maxSalary);  Iterable<Employee> searchByPosition(Iterable<Position> positions);  Iterable<Employee> allWithPositionAndMinSalary(Position position, **double** minSalary);  Iterable<Employee> searchByFirstName(String firstName);  Iterable<Employee> searchByNameAndPosition(String firstName, String lastName, Position position); } |

## Submission

Submit an archive (.zip) of the source code. Your code **mustn't** contain namespaces/packages.

## Scoring

Each implemented method brings you a specific amount of points, some of the points are awarded for correct behavior, others for performance. **The performance tests might not work on your PC**. You need to cover all tests in each group to receive points. Bellow is a breakdown of all points by methods:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Correct Behavior | Performance | Total |
| Overall | 40 | 60 | 100 |