# Data Structures Exam: Chainblock

## Description

Your good old friend Pesho (the big time capitalist) has invited you to join his team in the creation of a one of a kind database. You are ready to revolutionize the fintech industry beyond blockchains and payment solutions. You have the honor of taking the first step in the creation process. You need to implement a really fast data structure which will hold Transactions.

**Transaction** will hold:

* int Id – unique transaction id
* TransactionStatus Status – enumeration of work positions for the employee
* string From – the sender of the transaction
* string To – the receiver of the transaction
* double Amount – the salary of the employee

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

* Add() – Add a transaction to the record. You will need to implement the Contains() methods as well.
* Contains(Transaction) – checks if a given transaction is present in the record
* Contains(id) – checks if a transaction with the given id exists in the record
* Count – returns the number of transactions in the record
* ChangeTransactionStatus(id, status) – changes the status of the transaction with the given id or throws ArgumentException if no such transaction exists.
* RemoveTransactionById(id) – remove the transaction from the record if the id exists, otherwise throws InvalidOperationException
* GetById(id) – return the transaction with the given id. If such transaction doesn't exist, throw InvalidOperationException.
* GetByTransactionStatus(status) – return the transactions with the given status ordered by amount descending. If there are no transactions with the given status, throw InvalidOperationException
* GetAllSendersWithTransactionStatus(status) – returns all senders which have transactions with the given status. If no transactions exist, throw InvalidOperationException
* GetAllReceiversWithTransactionStatus(status) – returns all receivers which have transactions with the given status. Throw InvalidOperationException if no such transactions are present in the record
* GetAllOrderedByAmountDescendingThenById() – returns all transactions ordered by amount descending and by id
* GetBySenderOrderedByAmountDescending(sender) – search for all transactions with a specific sender and return them ordered by amount descending. If there are no such transactions throw InvalidOperationException
* GetByReceiverOrderedByAmountDescending(receiver) – returns all transactions with particular receiver ordered by amount descending. If there are no such transactions throw InvalidOperationException
* GetByTransactionStatusAndMaximumAmount(status, amount) – returns all transactions with given status and amount less or equal to a maximum allowed amount ordered by amount descending. Returns an empty collection if no such transactions were found
* GetBySenderAndMaximumAmountDescending(sender, amount) – returns all transactions with given sender and amount bigger than given amount ordered by amount descending. If there are no such transactions throw InvalidOperationException
* GetByReceiverAndAmountRange(receiver, lo, hi) – returns all transactions with given receiver and amount between lo (inclusive) and hi (exclusive) ordered by amount descending and by id. If there are no such transactions throw InvalidOperationException
* **GetAllInAmountRange(lo, hi)** –returns all transactions within a range by insertion order (the range is inclusive). Returns an empty collection if no such transactions were found.

The equivalent exception in java is IllegalArgumentException (all exceptions in java will be IllegalArgumentException).

## Input / Output

You are given a **Visual Studio C# project skeleton** (unfinished project) / **IntelliJ Java project** holding the interface IChainblock, the unfinished classes Chainblock and Transaction. **Tests** covering the Chainblock **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 Transaction class, but don't remove anything.
* You can edit the Chainblock class if it implements the IChainblock interface.

## Interface

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

|  |
| --- |
| public interface IChainblock : IEnumerable<Transaction>  {  int Count { get; }  void Add(Transaction tx);  bool Contains(Transaction tx);  bool Contains(int id);  void ChangeTransactionStatus(int id, TransactionStatus newStatus);  void RemoveTransactionById(int id);  Transaction GetById(int id);  IEnumerable<Transaction> GetByTransactionStatus(TransactionStatus status);  IEnumerable<Transaction> GetAllOrderedByAmountDescendingThenById();  IEnumerable<Transaction> GetBySenderOrderedByAmountDescending(string sender);  IEnumerable<Transaction> GetByReceiverOrderedByAmountThenById(string receiver);  IEnumerable<Transaction> GetByTransactionStatusAndMaximumAmount(TransactionStatus status, double amount);  IEnumerable<Transaction> GetBySenderAndMinimumAmountDescending(string sender, double amount);  IEnumerable<Transaction> GetByReceiverAndAmountRange(string receiver, double lo, double hi);  IEnumerable<Transaction> GetAllInAmountRange(double lo, double hi);  IEnumerable<string> GetAllSendersWithTransactionStatus(TransactionStatus status);  IEnumerable<string> GetAllReceiversWithTransactionStatus(TransactionStatus status);  } |

The interface IChainblock 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 |