# Data Structures Exam Retake: RoyaleArena

## Description

You have been invited by the blue king himself. He is in a huge need for an application to help him fight in the glorious Royale Arena. As one of the top players on the arena, he has a reputation to maintain, therefore he cannot afford to lose. The application must be able to keep and query all the cards that exist within the game. Each card has its own specifics. The trick is that the intervals between the battles are so small, that the application must be fast in order to help the king.

**Battlecard** will hold:

* int Id – unique card id
* CardType Type – enumeration of battlecard
* string Name – the name of the card
* int Damage – the damage of the card
* int Swag – the swag of the card

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

* Add() – Add a battle card to the arena. You will need to implement the Contains() methods as well. HashSet<Battlecard>
* Contains(Battlecard) – checks if a given battlecard is present in the arena. HashSet<Battlecard>
* Count – returns the number of cards in the arena
* ChangeCardType(id, type) – changes the status of the battlecard with the given id or throws ArgumentException if no such card exists. Dictionary<int,Battlecard>
* GetById(id) – return the card with the given id. If such card doesn't exist, throw InvalidOperationException. Dictionary<int,Battlecard>
* RemoveById(id) – Removes the card with the given id, otherwise throws InvalidOperationException Dictionary<int,Battlecard>
* GetByCardType(type) – Returns the cards with the given card type ordered by damage descending then by id. If there are no cards with the given type, throw InvalidOperationException Dictionary<CardType,OrderedDictionary<double,Battlecard>
* GetByTypeAndDamageRangeOrderedByDamageThenById(type, lo, hi) – returns all cards with particular card type ordered by damage descending (exclusive), then by id ascending. If there are no such types throw InvalidOperationException Dictionary<CardType,OrderedDictionary<double,Battlecard>
* GetByCardTypeAndMaximumDamage(type, damage) – returns all cards with given type and damage less or equal to a maximum allowed amount ordered by damage descending, then by id. Throws an InvalidOperationException if such cards were not found. Dictionary<CardType,OrderedDictionary<double,Battlecard>
* GetByNameOrderedBySwagDescending(name) – search for all cards with a specific name and return them ordered by swag descending then by id. If there are no such cards throw InvalidOperationException Dictionary<string,OrderedDictionary<double,Battlecard>
* GetByNameAndSwagRange(name, lo, hi) – returns all cards with given name and swag between lo (inclusive) and hi (exclusive) ordered by swag descending then by id. If there are no such cards throw InvalidOperationException. Dictionary<string,OrderedDictionary<double,Battlecard>

GetAllByNameAndSwag() – Returns the most swaggish card for each name. Throws InvalidOperationException/IllegalOperationException if none were found ???????

* FindFirstLeastSwag(n) – Returns the first n cards with least swag. If there are two identical swags, order by id. If the argument passed exceeds the count of the arena,

Throw InvalidOperationException/IllegalOperationException OrderedDictionary<double,Battlecard>

* **GetAllInSwagRange(lo, hi)** – returns all cards within a range ordered by swag(the range is inclusive). Returns an empty collection if no such cards were found. OrderedDictionary<double,Battlecard>
* **GetEnumerator()** – Iterate the arena by insertion order

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 IArena, the unfinished classes RoyaleArena and Battlecard. **Tests** covering the RoyaleArena **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 Battlecard class, but don't remove anything.
* You can edit the RoyaleArena class if it implements the IArena interface.

## Interface

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

|  |
| --- |
| public interface IArena : IEnumerable<Battlecard>  {  void Add(Battlecard card);  bool Contains(Battlecard card);  int Count { get; }  void ChangeCardType(int id, CardType type);  Battlecard GetById(int id);  void RemoveById(int id);  IEnumerable<Battlecard> GetByCardType(CardType type);  IEnumerable<Battlecard> GetByTypeAndDamageRangeOrderedByDamageThenById(CardType type);  IEnumerable<Battlecard> GetByCardTypeAndMaximumDamage(CardType type, int damage);  IEnumerable<Battlecard> GetByNameOrderedBySwagDescending(string name);  IEnumerable<Battlecard> GetByNameAndSwagRange(string name, int lo, int hi);  IEnumerable<Battlecard> GetAllByNameAndSwag(string name);  IEnumerable<Battlecard> FindFirstLeastSwag(int n);  IEnumerable<Battlecard> GetAllInSwagRange(int lo, int hi);  } |

The interface Arena in Java looks like the code below:

|  |
| --- |
| **public interface Arena extends Iterable<Battlecard> {**  **void add(Battlecard card);**  **boolean contains(Battlecard card);**  **int getCount();**  **void changeCardType(int id, CardType type);**  **Battlecard getById(int id);**  **void removeById(int id);**  **Iterable<Battlecard> GetByCardType(CardType type);**  **Iterable<Battlecard> GetByTypeAndDamageRangeOrderedByDamageThenById(CardType type);**  **Iterable<Battlecard> GetByCardTypeAndMaximumDamage(CardType type, int damage);**  **Iterable<Battlecard> GetByNameOrderedBySwagDescending(String name);**  **Iterable<Battlecard> GetByNameAndSwagRange(String name, int lo, int hi);**  **Iterable<Battlecard> GetAllByNameAndSwag(String name);**  **Iterable<Battlecard> FindFirstLeastSwag(int n);**  **Iterable<Battlecard> GetAllInSwagRange(int lo, int hi);**  **}** |

## 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 | 100 | 100 | 200 |