# Data Structures Exam: Hearthstone

## Description

Your task is to implement a simple game with cards. You may have already known about the **Hearthstone** game and now you must implement your custom version of it.

Level

 

Damage

Health

You have a class Card which has the following properties:

* String Name – unique name for each card
* Int Damage – the damage for the current card
* Int Score – the score current card has
* Int Health – the health of the current card (each card starts with 20 health)
* Int Level – the level of the card

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

* Draw(Card card)– Add a card to the deck of cards. You will need to implement the Contains() method as well. If the card name already exists throw ArgumentException
* Contains(name) – checks if a card with the given name is present in the deck
* Count – returns the number of cards in the deck
* Play(atackerCardName, atackedCardName) – find the attacker card and make damage to the attacked card. The attacker card can only attack other card if they both have the same level (otherwise throw ArgumentException). If the attacked card gets damage more than its current health it is no more available to play with, but it **doesn’t get removed from the deck**. If the attacker card kills the other card its score increases with the level of the attacked card. If the attacker tries to attack card with 0 or less health nothing happens. If any of the two provided cards does not exist throw ArgumentException
* Remove(name) – remove the card with the given name. If the card doesn’t exist return ArgumentException
* RemoveDeath() – remove all cards with health under or equal to 0
* GetBestOfRange(int start, int end) – find all cards which have a score between the given two inclusive and ordered by their level descending. If you don’t find any return empty collection.
* ListCardsByPrefix(prefix) – return all cards starting with the specified suffix (sorted by the ASCII code of the reversed name in ascending order as a first criteria and by level in ascending order as a second criteria).
* SearchByLevel(int)– return all cards with the given level, order them by score descending
* Heal(int health) – finds the card with the smallest health and increases it with the given health. Cards with negative health can get heal too.

*Feel free to override Equals() and GetHashCode() if necessary.*

## Input / Output

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

## Interface

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

|  |
| --- |
| public interface IBoard  {  void Draw(Card card);  bool Contains(string name);  int Count();  void Play(string attackerCardName, string attackedCardName);  void Remove(string name);  void RemoveDeath();  IEnumerable<Card> GetBestInRange(int start, int end);  IEnumerable<Card> ListCardsBySufficks(string sufficks);  IEnumerable<Card> SearchByLevel(int level);  void Heal(int health);  } |

The interface Board in Java looks like the code below:

|  |
| --- |
| **public interface Board {**  **void draw(Card card);   Boolean contains(String name);   int count();   void play(String attackerCardName, String attackedCardName);   void remove(String name);   void removeDeath();   Iterable<Card> getBestInRange(int start, int end);   Iterable<Card> listCardsBySufficks(String sufficks);   Iterable<Card> searchByLevel(int level);   void heal(int health); }** |

## 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 | 50 | 100 | 150 |