# Limited Memory – Data Structures Retake Exam

A **limited-memory collection** is a collection that cannot request more memory if its capacity is filled (unlike a list or a hash table which double their size when full). Instead, a limited-memory collection makes room for new elements by removing the **item requested the longest time ago.**

The collection is **generic** and stores **key-value pairs**. Here is how it works:

|  |
| --- |
| var collection = new LimitedMemoryCollection<string, int>(4);  collection.Set("Gosho", 5);  collection.Set("Penio", 3);  collection.Set("Prakash", 7);  collection.Set("Maria", 2); // Max capacity reached  collection.Set("Tanio", 3); // Removes Gosho to make room for Tanio  collection.Get("Penio");  collection.Set("Penka", 10); // Removes Prakash to make room for Penka  foreach (var record in collection)  {  Console.Write("{0}({1}) ", record.Key, record.Value);  // Maria(2) Tanio(3) Penio(3) Penka(10)  } |

Notice how calling **Get()** or **Set()** makes the record the **most recently requested**. In other words, when making room for a new record, we remove the record on which we called either **Get()** or **Set()** the longest time ago.

Your task is to design a data structure in C# or Java that supports the below-listed functionality in a **fast and efficient way**.

* **Get(key)** – returns the value that corresponds to the given key
  + Throws an exception if the key does not exist.
* **Set(key, value)** – sets a value to the corresponding key
  + If the key already exists, overwrites its value.
  + Adds the key if it does not exist. If the capacity is full, makes room for the new record by **removing the record requested the longest time ago.**
* **Capacity** – the maximum amount of records that can be stored in the collection
* **Count** – the current record count
* **GetEnumerator()** – retuns an enumerator for iterating over the elements. Starts from the **most recently requested**.

### Input and Output

You are given a **Visual Studio C# project skeleton** (unfinished project) / **IntelliJ Java project skeleton** (unfinished project) holding the interface ILimitedMemoryCollection, the unfinished class LimitedMemoryCollection<TKey,TValue> and **tests** covering its **functionality** and its **performance**.

|  |
| --- |
| **ILimitedMemoryCollection.cs** |
| public interface ILimitedMemoryCollection<K, V> : IEnumerable<Pair<K, V>>  {  int Capacity { get; }  int Count { get; }  void Set(K key, V value);  V Get(K key);  } |

Your task is to **finish the given LimitedMemoryCollection class** and make the tests run correctly.

* You are **not allowed to change the tests**.
* You are **not allowed to change the interface**.

### Constraints

* **Capacity** will be in the range [4...200000].

### Submissions

Submit an archive (.zip) of the source code + external libraries.

# Scoring

Each implemented method brings you a specific amount of points, some of the points are awarded for correct behavior, others for performance. Performance tests are directly dependent on correctness, if a method does not work correctly you will not receive points for performance. You need to cover all tests in a given group in order to receive points. Bellow is a breakdown of all points by methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Correct Behaviour | Performance | Total |
| Get | 8 | 15 | 21 |
| Set | 12 | 20 | 32 |
| ForEach | 10 | 15 | 23 |
| Overall: | 30 | 50 | 80 |