**Data Structures – Fundamentals – Exam Preparation**

This document defines the examination example problems for ["Data Structures – Fundamentals (C#)" course @ Software University](https://softuni.bg/trainings/3112/data-structures-fundamentals-with-csharp-september-2020).

Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/2554/07-Exam-Preparation).

Any code files that are part of the task are provided as **Skeleton**. In the beginning import the project skeleton, do not change any of the interfaces or classes provided. You are free to add additional logic in form of methods in both interfaces and implementations you are not allowed to delete or remove any of the code provided. Do not change the names of the files as they are part of the tests logic. **Do not change the namespaces** or move any of the files provided inside the skeleton if you have to add new file add it in the same class library.

Some **tests may be provided** within the skeleton – use those for local **testing and debugging**, however there **is no guarantee that there are no hidden tests added inside Judge**.

Please follow the exact instructions on uploading the solutions for each task. Submit as **.zip archive** the files contained inside **the library** folder (everything **except** the content inside **bin** and **obj** folders) this should work for all tasks regardless of current DS implementation.

You have to **study** the provided **skeleton**. The code is **separated** inside **different** **libraries**.

There **are** **few** **entities** **inside** the **project** you are **allowed** to **add** code to those, for example **Equals()** and **GetHashCode()** etc…

All the **entities** **objects** are **stored, ordered, created and used** inside the tests by their **IDs**. This means that the **IDs** will **always be unique** in **increasing order** and **non-negative** integers. You can use that for **priority, search, object identification** and all the operations that **require** to **map** an **object** to exact **value**.

**In addition, all the entities have parents and children try to use those fields if you need them to build some kind of hierarchy.**

# Loader

You are given a skeleton with a class **Loader** that implements the **IBuffer interface.** The methods **are not** **implemented** your task is to **implement them:**

* **void** **Add (IEntity entity)** – **adds** an entity inside the **loader**
* **IEntity Extract (int id) – removes and returns the entity** with the given **id.** If the entity is **not present** or the **Loader is empty** return **null**
* **IEntity Find (IEntity entity) – finds and returns the entity.** If the entity is **not present or the Loader is empty return null**
* **bool Contains(IEntity entity) –** returns **true** if the entity is stored and **false otherwise**
* **int EntitiesCount –** returnsthe **number** of entities stored
* **void Replace (IEntity oldEntity, IEntity newEntity) – finds** and replaces the first parameter (oldEntity) with the second one (newEntity) **if present.** If **not** throw **InvalidOperationException("Entity not found")**
* **void Swap (IEntity first, IEntity second) – finds and swaps** the first parameter (first) with the second one (second) **if both are present.** If **not** throw **InvalidOperationException("Entity not found")**
* **void Clear ()** **–** **clears all the entities stored** the Loader is left empty
* **IEntity[] ToArray ()** – returns all the elements **as an array**, if the Loader is empty return an **empty array**
* **List<IEntity> RetainAllFromTo** **(BaseEntityStatus lowerBound, BaseEntityStatus upperBound) –** returnsall the entities with given status **within the range (both are inclusive)** if none are found returnan **empty List**
* **List<IEntity> GetAll () – returns all the entities.** If none returns an **empty List**
* **void UpdateAll (BaseEntityStatus oldStatus, BaseEntityStatus newStatus) – updates all** the entities with the given status (oldStatus) **and sets** their status to (newStatus)
* **void RemoveSold () –** removes **all the entities** with status **"SOLD"** from the Loader
* **IEnumerator<IEntity> GetEnumerator() –** iterates through the collection

1. **Performance Tests - Loader**

For this task you will only be required to submit the **code from the Loader problem**. If you are having problem with this task you should **perform detailed algorithmic complexity analysis** and try to **figure** **out** **weak** spots inside your implementation.

**For this problem it is important that other operations are implemented correctly according to the specific problems: Add (), Size (), Count () etc…**

1. **Data**

You are given a skeleton with a class **Data** that implements the **Repository interface.** The methods **are not** **implemented** your task is to **implement them:**

* **int Size –** returnsthe **number** of entities stored
* **void Add(IEntity entity) – adds** an entity inside the Data
* **IEntity GetById (int id) –** returnsanentity by the id if the id is invalid **return null**
* **List<IEntity> GetByParentId (int id) –** returnsall the entities by the **id of their parent if the id is invalid return empty List**
* **List<IEntity> GetAll () –** returns all the entities if there are none return an **empty List**
* **IRepository Copy () –** returnsexact copy of the data object
* **List<IEntity> GetAllByType (string type) –** returns **all the entities** by given type if there are none return an empty list. The parameter is a string which is the **type. If the parameter is invalid throw InvalidOperationException ("Invalid type: " + type)**
* **IEntity PeekMostRecent () –** returnsthe most recent object **(the object with greatest ID)** added **without** **removing** it, the criteria is the **id** of the object if there are no elements **throw InvalidOperationException ("Operation on empty Data")**
* **IEntity DequeueMostRecent () –** returnsand **removes** the most recent object **(the object with greatest ID)** added without removing it, the criteria is the **id** of the object if there are no elements throw **InvalidOperationException ("Operation on empty Data")**

1. **Performance Tests - Data**

For this task you will only be required to submit the **code from the Data problem**. If you are having problem with this task you should **perform detailed algorithmic complexity analysis** and try to **figure** **out** **weak** spots inside your implementation.

**For this problem it is important that other operations are implemented correctly according to the specific problems: Add (), Size (), Count () etc…**