**Data Structures – Fundamentals – Exam Preparation**

This document defines the examination example problems for ["Data Structures – Fundamentals (Java)" course @ Software University](https://softuni.bg/trainings/2812/data-structures-fundamentals-with-java-march-2020).

Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/2292/07-Exam-Preparation).

Write Java code for solving the tasks on the following pages. Code should compile under the Java 8 and above standards you can write and locally test your solution with the Java 13 standard, however **Judge will run the submission with Java 10 JRE**. Avoid submissions with **features included after Java 10** release doing **otherwise** will result in **compile time error**.

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 packages** or move any of the files provided inside the skeleton if you have to add new file add it in the same package of usage.

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 **"...\src\main\java"** folder this should work for all tasks regardless of current DS implementation.

In order for the solution to compile the tests **successfully** the project **must** have **single** **Main.java** file containing single **public static void main(String[] args)** method even empty one within the **Main class**.

You have to **study** the provided **skeleton**. The code is **separated** inside **different** **packages**, for you tasks you should be writing code **mainly** **inside** **the "core" package**.

There **are** **few** **entities** **inside** the **project** you are **allowed** to **add** code to those, for example **equals** **()** and **hashCode** **()** 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 **Buffer interface.** The methods **are not** **implemented** your task is to **implement them:**

* **void** **add (Entity entity)** – **adds** an entity inside the **loader**
* **Entity 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**
* **Entity find (Entity entity) – finds and returns the entity.** If the entity is **not present or the Loader is empty return null**
* **boolean contains(Entity entity) –** returns **true** if the entity is stored and **false otherwise**
* **int entitiesCount () –** returnsthe **number** of entities stored
* **void replace (Entity oldEntity, Entity newEntity) – finds** and replaces the first parameter (oldEntity) with the second one (newEntity) **if present if not throw**  ("Entity not found")
* **void swap (Entity first, Entity second) – finds and swaps** the first parameter (first) with the second one (second) **if both are present if not** throw IllegalStateException("Entities not found")
* **void clear ()** **–** **clears all the entities stored** the Loader is left empty
* **Entity[] toArray ()** – returns all the elements **as an array**, if the Loader is empty return an **empty array**
* **List<Entity> retainAllFromTo** **(Status lowerBound, Status upperBound) –** returnsall the entities with given status **within the range (both are inclusive)** if none are found returnan **empty List**
* **List<Entity> getAll () – returns all the entities** if none returns an **empty List**
* **void updateAll (Status oldStatus, Status 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

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:**

* **void add(Entity entity) – adds** an entity inside the Data
* **Entity getById (int id) –** returnsanentity by the id if the id is invalid **return null**
* **List<Entity> getByParentId (int id) –** returnsall the entities by the **id of their parent if the id is invalid return empty List**
* **List<Entity> getAll () –** returns all the entities if there are none return an **empty List**
* **Repository copy () –** returnsexact copy of the data object
* **List<Entity> 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 llegalArgumentException ("Illegal type: " + type)**
* **Entity 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 IllegalStateException("Operation on empty Data")**
* **Entity pollMostRecent () –** 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 **IllegalStateException("Operation on empty Data")**
* **int size () –** returnsthe **number** of entities stored

## Performance Tests

For this task you will only be required to submit the **code from the previous two problems**. First **half** of the tests will **test the performance of the first task**, the **second** one will tests the **second**. 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 (), Iterable () etc…**