**Data Structures Advanced Retake Exam – Java**

This document defines the exam preparation for ["Data Structures – Advanced (Java)" course @ Software University](https://softuni.bg/trainings/3418/data-structures-advanced-with-java-august-2021).

Please submit your solutions (**source code**) of all below described problems in [Judge](https://judge.softuni.org/Contests/3233/Data-Structures-Advanced-with-Java-Retake-Exam-26-Sept-2021).

# Movie Database – 100 pts

Movie Database is an interactive movie database, which provides functionality for registering movies and manipulating them in a repository-like structure.

You are given a skeleton with a class **MovieDatabaseImpl** that implements the **MovieDatabase interface.**

This **MovieDatabase** works with **Movie** entities. All **Movie** entities are identified by a **unique Id**.

The **Movie** entity contains the following properties:

* **Id** – string
* **Name** – string
* **ReleaseYear** – integer
* **Rating** – double
* **Actors** – list of strings

Implement the following functionalities to make the **MovieDatabase** fully operative:

* **void addMovie(Movie movie)** – **adds** a **movie** to the **MovieDatabase**.
* **void removeMovie(String movieId)** – **removes** the **movie** with the given **id.**   
  If there is no such **movie** - **throw IllegalArgumentException()**
* **bool contains(Movie movie)** –returns whether the movie is **contained** inside the **MovieDatabase**.
* **int size() –** returns the **total count** of all **movies**.
* **Iterable<Movie> getMoviesByActor(String actorName)** –returns **all movies**, which have the **actor** with the **given name**, **ordered** by **rating** in **descending order** and by **release year** in **descending order**.   
  If there is are **no movies** that **contain** the **given actor** - **throw IllegalArgumentException()**
* **Iterable<Movie> getMoviesByActors(List<String> actors)** –returns **all movies**, which contain **ALL of the given actors**, **ordered** by **rating** in **descending order** and by **release year** in **descending order**.   
  If there is are **no movies** that **contain ALL of the given actor** - **throw IllegalArgumentException()**
* **Iterable<Movie> getMoviesByYear(Integer year)** –returns **all movies**, which have a **release year** equal to the **given year**, **ordered** by **rating** in **descending order**. If there aren’t any moviess – return an **empty collection**.
* **Iterable<Movie> getMoviesInRatingRange(int lowerBound, int upperBound)** – returns all of the **movies** with **rating** in the range specified with **lower bound** and **upper bound.** Both bounds are **inclusive**.The results should be ordered by **rating** in **descending order**. If there aren’t any mvoiess in the specified range – return an **empty collection**.
* **Iterable<Movie> getAllMoviesOrderedByActorPopularityThenByRatingThenByYear()** – returns all of the **movies** ordered by **total amount of movies** in which **their** **actors played** in **descending order**, then by **rating** in **desceding order** and then by **release year** in **descending order**. If there aren’t any movies – return an **empty collection**. **Explanation of sorting criteria:** Let’s say a movie has actors – **John**, **Jack**, **Joseph**.
  + **John** played in this movie and **2** other movies **(3)**
  + **Jack** in this movie and **1** other movies **(2)**
  + **Joseph** only plays in **this** **movie (1)**

This movie’s **total** **amount of movies** in **which it’s actors** **played** is **6**.

**NOTE: If all sorting criteria fails, you should order by order of input. This is for all methods with ordered output.**

* 1. **Movie Database – Performance – 50 pts**

For this task you will only be required to submit the **code from the previous problem**. If you are having a 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**, **remove**, **get** etc… Also, make sure you are using the correct data structures. ☺

You can submit code to this problem **without full coverage** from the previous problem, **not all test cases** will be considered, only the **general** **behaviour** will be important, **edge** **cases** will mostly be ignored such as throwing exceptions etc…

# Doodle – 100 pts

Google has bankrupted and is closing doors. The world is in a predicament. This is the ideal moment to provide an alternative. Our good old friend Melon Usk has decided to take the initiative. Guess who’s gonna help him once more. That’s right – you!

Doodle Search is a Search engine platform, which works with entities called Doodles. The data you will be working with contains very basic information about the Doodles – ids, titles, visits etc. There are also Doodles which are Ads. Normally all Doodles generate revenue, but the Ads are special, they generate Revenue for every visit they get.

You are given a skeleton with a class **DoodleSearchImpl** that implements the **DoodleSearch interface.**

This **DoodleSearch** works with **Doodle** entities. All **Doodle** entities are identified by a **unique Id**.

The **Doodle** entity contains the following properties:

* **Id** – string
* **Title** – string
* **Visits** – int
* **IsAd** – bool
* **Revenue –** double

Implement the following functionalities to make the **DoodleSearch** fully operative:

* **void addDoodle(Doodle doodle)** – **adds** a **Doodle** to the **DoodleSearch**. **NOTE**: Doodle titles are **unique**.
* **void removeDoodle(String doodleId)** – **removes** the **doodle** with the given **id** from the **DoodleSearch**.If there is no such **doodle** - **throw IllegalArgumentException()**
* **bool contains(Doodle doodle)** –returns whether the doodle is **contained** inside the **DoodleSearch**.
* **int size() –** returns the **total count** of all **doodles**.
* **Doodle getDoodle(String doodleId)** – **returns** the **doodle** with the given **id**.If there is no such **doodle** - **throw IllegalArgumentException()**
* **double getTotalRevenueFromDoodleAds()** – **returns** the **total revenue** from all **Doodles** that are **ads**. The total revenue is calculated by **multiplying** the **visits** by the **revenue** for each ad **Doodle** (**visits \* revenue**). You need to calculate that for all **Doodles** that **ARE ADS**, then you need to **sum** the **values** and return the result.
* **void visitDoodle(String doodleTitle)** – **increases** the **visits** of the **Doodle** with the **given title** with **1**. If there is no such **doodle** - **throw IllegalArgumentException()**
* **Iterable<Doodle> searchDoodles(string searchQuery) –** returns **all doodles** that have a **title**, which **contains** the **search query**. The results should be **ordered** by **relevance (earliest match of the search query)**, then by **visits** in **descending order**. but **Ad Doodles** should **come first**, regardless of **relevance** or **visits**.
  + **NOTE**: **Ad** **Doodles** should also be **ordered** by **relevance** and by **visits** in **descending order**.
  + If the search query is "**asd**" and we have titles "**casd**" and "**ddasd**", then "**casd**" comes **first** (more **relevant**). (other example: "**casd**" and "**basd**" are equal)
  + If there aren’t any doodles that match the search query – return an **empty collection**.
* **Iterable<Doodle> getDoodleAds()** –returns **all Doodles** that are **Ads**. The results should be ordered by **revenue** in **descending order** and then by **visits** in **descending order**.
  + If there aren’t any doodle ads – return an **empty collection**.
* **Iterable<Doodle> getTop3DoodlesByRevenueThenByVisits()** – returns the **top 3** of the **Doodles** in terms of **revenue** in **descending order**, then by **visits** in **descending** order. If there aren’t any doodles – return an **empty collection**.

**NOTE: If all sorting criteria fails, you should order by order of input. This is for all methods with ordered output.**

* 1. **Doodle – Performance – 50 pts**

For this task you will only be required to submit the **code from the previous problem**. If you are having a 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**, **remove**, **get** etc… Also, make sure you are using the correct data structures. ☺

You can submit code to this problem **without full coverage** from the previous problem, **not all test cases** will be considered, only the **general** **behaviour** will be important, **edge** **cases** will mostly be ignored such as throwing exceptions etc…