**Data Structures Advanced Regular Exam – Java**

This document defines the exam preparation for "Data Structures – Advanced (Java)" course @ Software University.

Please submit your solutions (**source code**) of all below described problems in Judge.

# Discord – 100 pts

Discord is a simple messaging repository, allowing storage of messages in chat channels.

You are given a skeleton with a class **DiscordImpl** that implements the **Discord interface.**

This **Discord** works with **Message** entities. All **Message** entities are identified by a **unique Id**.

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

* **Id** – string
* **Content** – string
* **Timestamp** – integer
* **Channel** – string
* **Reactions** – List of strings

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

* **void sendMessage(Message message)** – **adds** a **message** to **Discord**.
* **bool contains(Message message)** –returns whether the message is **contained** inside **Discord**.
* **int size() –** returns the **total count** of all **messages**.
* **Message getMessage(String messageId)** – **retrieves** the **message** with the given **id.**   
  If there is no such **message** - **throw IllegalArgumentException()**
* **void deleteMessage(String messageId)** – **removes** the **message** with the given **id.**   
  If there is no such **message** - **throw IllegalArgumentException()**
* **void reactToMessage(String messageId, String reaction)** – **adds** the **given reaction** to the **message** with the given **id**  
  If there is no such **message** - **throw IllegalArgumentException()**
* **Iterable<Message> getChannelMessages(String channel)** –returns **all messages**, which are in the **given channel.**  
  If there are **no messages** in the **given channel** - **throw IllegalArgumentException()**
* **Iterable<Message> getMessagesByReactions(List<String> reactions)** –returns **all messages**, which contain **ALL of the given reactions**, **ordered** by **count of reactions** in **descending order** and by **timestamp** in **ascending order**.
* **Iterable<Message> getMessagesInTimeRange(int lowerBound, int upperBound)** – returns all of the **messages** with **timestamp** in the range specified with **lower bound** and **upper bound.** Both bounds are **inclusive**.The results should be ordered by **count** of **total messages contained** in **each** **message**’s **channel**, in **descending order**. If there aren’t any messages in the specified range – return an **empty collection**.
* **Iterable<Message> getTop3MostReactedMessages()** – returns the top 3 messages in terms of count of reactions in **descending order.** If there aren’t any messages – return an **empty collection**.
* **Iterable<Message> getAllMessagesOrderedByCountOfReactionsThenByTimestampThenByLengthOfContent()** – returns all of the **messages** ordered by **count of reactions** in **descending order**, then by **timestamp** in **ascending** **order** and then by **length of content** in **ascending order**. If there aren’t any messages – 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. **Discord – 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…

# MoovIt – 100 pts

MoovIt is a simple maps application which helps people pick a route to a designated point. You have been tasked with implementing the shell for the application logic – which is the component used for storing and managing routes.

Clients of the MoovIt app also have favorite routes – which always appear pinned to the top.

**NOTE**: Location points should always be considered in the terms of their logical order. For a single route, with 5 points, the point with index 0 is a starting point and the point with index 4 is the end point. We go from 0 – 4.

**NOTE**: **MoovIt** tends to filter duplicate routes. If 2 routes have the same **starting point** (index 0) and **ending point** (last index) and they have the **same distance** – they should be considered the **same route**.

You are given a skeleton with a class **MoovItImpl** that implements the **MoovIt interface.**

This **MoovIt** works with **Route** entities. All **Route** entities are identified by a **unique Id**.

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

* **Id** – string
* **Distance** – double
* **Popularity** – int
* **IsFavorite** – bool
* **LocationPoints –** List of strings

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

* **void addRoute(Route route)** – **adds** a **Route** to **MoovIt**.  
  If **route** already exists- **throw IllegalArgumentException()**
* **void removeRoute(String routeId)** – **removes** the **route** with the given **id** from **MoovIt**.If there is no such **route** - **throw IllegalArgumentException()**
* **bool contains(Route route)** –returns whether the route is **contained** inside **MoovIt**.
* **int size() –** returns the **total count** of all **routes**.
* **Route getRoute(String routeId)** – **returns** the **route** with the given **id**.If there is no such **route** - **throw IllegalArgumentException()**
* **void chooseRoute(String routeId)** – **increases** the **popularity** of the **Route** with the **given id** with **1**. If there is no such **route** - **throw IllegalArgumentException()**
* **Iterable<Route> searchRoutes(String startPoint, String endPoint) –** returns **all routes** that have alogical route, which **contains** the **starting point** and **the end point**. The results should be **ordered** by **distance between the 2 points (lowest count of points between them)**, then by **popularity** in **descending order**. but **Favourity Routes** should **come first**, regardless of **distance** or **popularity**.
  + **NOTE**: **Favourity Routes** should also be **ordered** by **distance** and by **popularity** in **descending order**.
  + **Explanation**: If the given points are **Sofia** (start) and **Plovdiv** (end) and we have the following routes:
    - **Route 1** **–** LocationPoints (Vraca -> Sofia -> Ihtiman -> Pazardzhik -> Plovdiv)
    - **Route 2 –** LocationPoints (Pleven -> Sofia -> Pazardzhik -> Plovdiv)
    - **Route 3 –** LocationPoints (Belgrade -> Sofia -> Plovdiv)
    - The order by distance would be **Route 3 -> Route 2 -> Route 1**, because the **number** of **locations** between the desired ones is **lowest** at **Route 3**.
  + If there aren’t any routes that match the search points – return an **empty collection**.
* **Iterable<Route> getFavoriteRoutes(String destinationPoint)** –returns **all Routes** that are **Favorite** and contain the given **destinationPoint** as a non-starting point (not first index). The results should be ordered by **distance** in **ascending order** and then by **popularity** in **descending order**.
  + If there aren’t any favorite routes – return an **empty collection**.
* **Iterable<Route> getTop5RoutesByPopularityThenByDistanceThenByCountOfLocationPoints()** – returns the **top 5** of the **Routes** in terms of **popularity** in **descending order**, then by **Distance** in **ascending** order and then by **count** of **location points** in **ascending order**. If there aren’t any routes – 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. **MoovIt – 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…