Data Structures Advanced with Java - Regular Exam

1. Task Manager – 100 pts

The Task Manager is a simple system which stores executable tasks orriented towards specific logical domains.

You are given a skeleton with a class **TaskManagerImpl** that implements the **TaskManager interface.**

This **TaskManager** works with **Task** entities. All **Task** entities are identified by a **unique Id**.

NOTE: The TaskManager should store **executable tasks** and **executed tasks** also (as they can be **rescheduled**)

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

- **Id** – String
- **Name** – String
- **Estimated Execution Time (EET)** – integer
- **Domain** – String

Implement the following functionalities to make **Task Manager** fully operative:

- **void addTask(Task task)** – **adds** a **task** pending execution to the **Task Manager**. Tasks should be stored in order of addition.
- **bool contains(Task task)** – returns whether the task is **contained** inside the **Task Manager**.
- **int size() –** returns the **total count** of all unexecuted **tasks**.
- **Task getTask(String taskId)** – **retrieves** the **task** with the given **id.**
  If there is no such **task** - **`throw IllegalArgumentException()`**
- **void deleteTask(String taskId)** – **deletes** the **task (completely – removes any reference to it from the Task Manager)** with the given **id.**
  If there is no such **task** - **`throw IllegalArgumentException()`**
- **Task executeTask()** – **executes** the **first task (first added)** in the execution queue**, removing** it from the **execution queue** and saving it as an **executed** task, returns the **Task** as a result.
  If there is no such **task** - **`throw IllegalArgumentException()`**
- **void rescheduleTask(String taskId)** – **reschedules** an **executed task** with the given id**,** adding it once more to the **Task Manager's** execution queue.
  If there is no such **executed task** - **`throw IllegalArgumentException()`**
- **Iterable<Task> getDomainTasks(String domain)** – returns **all unexecuted tasks**, which are in the **given domain.**
  If there are **no tasks** in the **given domain** - **`throw IllegalArgumentException()`**
- **Iterable<Task> getTasksInEETRange(int lowerBound, int upperBound)** – returns all of the **unexecuted tasks** with **an EET** in the range specified with **lower bound** and **upper bound.** Both bounds are **inclusive**. The results should be ordered by **position** in the **execution queue**, in **ascending order**. If there aren't any tasks in the specified range – return an **empty collection**.

- **Iterable<Task> getAllTasksOrderedByEETThenByName()** – returns **all** of the **tasks** ordered by **EET** in **descending order**, then by **length** of **name** in **ascending order**

  If there aren't any tasks – 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.

1.5 Task Manager – 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…

2. Categorization – 100 pts

You have been tasked with creating a categorization infrastructure for a large store provider. The Categorization structure works with categories as its main entities, but it has quite the exciting functional requirements. Categories can be parents or children of other categories.

You are given a skeleton with a class **CategorizatorImpl** that implements the **Categorizator interface.**

This **Categorizator** works with **Category** entities. All **Category** entities are identified by a **unique Id**.

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

- **Id** – string
- **Name** – string
- **Description** – string

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

- **void addCategory(Category category)** – **adds** a **Category** to the **Categorizator**.
  If the **category** already exists - `throw IllegalArgumentException()`
- **void assignParent(String childCategoryId, String parentCategoryId)** – **adds** the **category** with the **given child id** as a **child** of the **category** with the **given parent id**.
  - **NOTE:** Categories will always have **only 1 parent**. There's no need for extra validation for that.

  If the either one of the given **categories** does not exist - `throw IllegalArgumentException()`

  If the **child category already is** a **child** of the **parent category** - `throw IllegalArgumentException()`
- **void removeCategory(String categoryId)** – **removes** the **category** with the given **id** from the **Categorizator**. This also **removes** all **child categories** associated with it.
  If there is no such **category** - `throw IllegalArgumentException()`
- **bool contains(Category category)** – returns whether the category is **contained** inside the **Categorizator**.
- **int size() –** returns the **total count** of all **categories**.

- **Iterable<Category> getChildren(String categoryId)** – **returns all child (direct + indirect) categories** of the **category** with the given **id**.
  If there is no such **category** - **`throw IllegalArgumentException()`**
- **Iterable<Category> getHierarchy(String categoryId)** – returns **all parents** of the **category** (and the category itself) with the **given id** in sequential order in terms of their hierarchical level.
  - **Explanation**: If A is parent of B**,** and B is parent of C, and C is parent of D -> if we request the hierarchy of category D, we should receive a collection containing [A, B, C, D] in this order.

  If there is no such **category** - **`throw IllegalArgumentException()`**

- **Iterable<Category> getTop3CategoriesOrderedByDepthOfChildrenThenByName()** – returns the **top 3** of the **Categories** ordered in terms of **depth** of **child categories** beneath them (how **deep** do their **children** go in terms of **hierarchical level**), then by **name** in **alphabetical (ascending) order**. If there aren't any categories – 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.

2.5 Categorization – 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…