**Data Structures Fundamentals –**

**Retake Exam – C#**

# ViTube – 100 pts

You’ve been tasked with implementing the data management of an online video platform. The software should work with users, which upload and watch videos.

You are given a skeleton with a class **ViTubeRepository** that implements the I**ViTubeRepository interface.**

**ViTube** works with **Users** and **Videos** asentities. All entities are identified by a **unique Id**.

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

* **Id** – string
* **Username** – string

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

* **Id** – string
* **Title** – string
* **Length** – double
* **Views** – Integer
* **Likes**– Integer
* **Dislikes** – Integer

Implement the following functionalities to make the **ViTube** software fully operative:

* **void RegisterUser(User user)** – **adds** an **user** to the **ViTube** software.
* **void PostVideo(Video video)** – **adds** a **video** to the **ViTube** software.
* **bool Contains(User user)** –returns whether the **user** is **contained** inside the **ViTube** software.
* **bool Contains(Video video)** –returns whether the **video** is **contained** inside the **ViTube** software.
* **IEnumerable<Video> GetVideos() –** returns a collection of all **videos**.
* **void WatchVideo(User user, Video video) –** the given **user**, watches the given **video** - **incrementing** the **views** of the **given video** with **1**. If either the user or video are not contained in the **ViTube** - **throw ArgumentException()**
* **void LikeVideo(User user, Video video) –** the given **user**, likes the given **video** - **incrementing** the **likes** of the **given video** with **1**. If either the user or video are not contained in the **ViTube** - **throw ArgumentException()**
* **void DislikeVideo(User user, Video video) –** the given **user**, dislikes the given **video** - **incrementing** the **dislikes** of the **given video** with **1**. If either the user or video are not contained in the **ViTube** - **throw ArgumentException()**
* **IEnumerable<User> GetPassiveUsers() –** returns a collection of all **users**, which have never **watched**, **liked** or **disliked** a video.
* **IEnumerable<Video> GetVideosOrderedByViewsThenByLikesThenByDislikes()** – returns all of the **videos** ordered by **views** in **descending order**, then by **likes** in **descending order**, then by **dislikes** in **ascending order**.   
  If there aren’t any videos – return an **empty collection**.
* **IEnumerable<User> GetUsersByActivityThenByName()** – returns all of the **users** ordered by **count** of **videos** they’ve **watched** in **descending order**, then by **count** of **videos** they’ve **liked** or **disliked** in **descending** order, and lastly – by **username** in **alphabetical (ascending)** **order**.   
  If there aren’t any users – 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. **ViTube – 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…

# Movie Database – 100 pts

You’ve been tasked with implementing a program for managing a movie database. The software should work with actors and movies.

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

This **MovieDatabase** works with **Actors** and **Movies** asentities. All entities are identified by a **unique Id**.

**NOTE**: Two actors could have the same movie.

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

* **Id** – string
* **Name** – string
* **Age** – integer

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

* **Id** – string
* **DurationInMinutes** – integer
* **Title** – string
* **Rating** – double
* **Budget**– double

Implement the following functionalities to make the **Movie Database** software fully operative:

* **void AddActor(Actor actor)** – **adds** an **actor** to the **Movie Database** software.
* **void AddMovie(Actor actor, Movie movie)** – **adds** a **movie** to the given **actor** in the **Movie Database** software. If the actor does not exist - **throw ArgumentException()**
* **bool Contains(Actor actor)** –returns whether the **actor** is **contained** inside the **Movie Database** software.
* **bool Contains(Movie movie)** –returns whether the **movie** is **contained** inside the **Movie Database** software.
* **IEnumerable<Movie> GetAllMovies() –** returns a collection of all **movies**.
* **IEnumerable<Actor> GetNewbieActors() –** returns a collection of all **actors** that **do not have any movies**.
* **IEnumerable<Movie> GetMoviesOrderedByBudgetThenByRating()** – returns all of the **movies** ordered by **budget** in **descending order**, then by **rating** in **descending** **order**.   
  If there aren’t any movies – return an **empty collection**.
* **IEnumerable<Actor> GetActorsOrderedByMaxMovieBudgetThenByMoviesCount()** – returns all of the **actors** ordered by **maximum budget of one of their movies** in **descending order**, then by **count** of **movies** in **descending** **order**.

If there aren’t any actors – return an **empty collection**.

* **IEnumerable<Movie> GetMoviesInRangeOfBudget(double lower, double upper)** – returns all of the **movies** ordered by **rating** in **descending order**, which have a **budget** in the **range** defined by the **given** **lower** and **upper boundaries**. The range is **inclusive**.   
  If there aren’t any movies – 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. **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.