**Data Structures Fundamentals – Exam**

# **Public Transport Management System**

**Correctness - 100 pts**

You are given a skeleton with a class **PublicTransportRepository** that implements the **IPublicTransportRepository** interface.

The system works with **Passenger** and **Bus** entities. All entities are identified by a **unique Id**.

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

* **Id** – string
* **Name** – string

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

* **Id** – string
* **Number** – string
* **Capacity** – Integer

Implement the following functionalities to make the **Public Transport Management System** fully operative:

* **void RegisterPassenger(Passenger passenger)** - adds a passenger to the system.
* **void AddBus(Bus bus)** - adds a bus to the system**.**
* **bool Contains(Passenger passenger)** - returns whether the passenger is in the system.
* **bool Contains(Bus bus)** - returns whether the bus is in the system.
* **IEnumerable<Bus> GetBuses()** - returns a collection of all buses in the system.
* **void BoardBus(Passenger passenger, Bus bus)** - the given passenger boards the given bus. If either the **passenger** or **bus** **is not in the system** - **throw ArgumentException().**
* **void LeaveBus(Passenger passenger, Bus bus)** - the given passenger leaves the given bus. If either the **passenger** or **bus is not in the system** or **the passenger is not on the bus** - **throw ArgumentException()**.
* **IEnumerable<Passenger> GetPassengersOnBus(Bus bus)** - returns a collection of all passengers currently on the given bus.
* **IEnumerable<Bus> GetBusesOrderedByOccupancy()** - returns all buses ordered by the **number of passengers currently on board** in ascending order.
* **IEnumerable<Bus> GetBusesWithCapacity(int capacity)** - returns a collection of all buses with a **capacity greater than or equal** to the given capacity.

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

**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** **behavior** will be important, and **edge** **cases** will mostly be ignored such as throwing exceptions, etc…