**Data Structures Java Exam Retake**

This document defines the examination problems for ["Data Structures – Fundamentals (Java)" course @ Software University](https://softuni.bg/trainings/3922/data-structures-fundamentals-with-java-november-2022).

Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.org/Contests/3416/Data-Structures-Fundamentals-with-Java-Retake-Exam-27-March-2022).

**Do not modify the interface or the package, or anything from the given recourses. In Judge you only upload the archive of the corresponding package.**

**Graphical user interface, text

Description automatically generatedText

Description automatically generated**

# **Barber Shop**

You are given a skeleton with a class **BarberShopImpl** that implements the **BarberShop interface.**

The BarberShopImpl works with **Barber** & **Client entities**, **all** **barbers and clients** are identified by their **unique** **names**. Implements all the operations from the **interface:**

* **void addBarber(Barber b)** – **adds** a barber. If there **is** **a barber** **with** **the** **same** **name** added before, throw IllegalArgumentException.
* **void addClient(Client c)** – **adds a client**. If a client with the same name exists, throw IllegalArgumentException.
* **boolean exist(Barber b) –** returns whether the **Barber** has been **added** or **not.**
* **boolean exist(Client c)** **–** returns whether the **Client** has been **added** or **not.**
* **Collection<Barber> getBarbers() –** returns all added barbers. If there aren’t any - return empty collection.
* **Collection<Client> getClients() –** returns all added clients. If there aren’t any - return empty collection.
* **void assignClient(Barber b, Client c)** – **adds a client for** the provided **barber**. If the barber or the client does not exist, throw IllegalArgumentException.
* **void deleteAllClientsFrom(Barber b) –** **Deletes** all assigned clients for the provided **barber.** If the barber does not exist throw IllegalArgumentException.
* **Collection<Client> getClientsWithNoBarber()** – **return only** clients with no assigned barber.
* **Collection<Barber> getAllBarbersSortedWithClientsCountDesc()** – **return all added barbers** ordered by their clients count descending. If there are not any barbers return empty collection.
* **Collection<Barber> getAllBarbersSortedWithStarsDecsendingAndHaircutPriceAsc()** – returns all **barbers** sorted by their stars descending and their haircut price ascending.
* **Collection<Client> getClientsSortedByAgeDescAndBarbersStarsDesc()** – return only clients who are assigned to barber and sorted by their age descending and by their barber stars descending**.**

# **Barber Shop – Performance**

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…

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…

# **Trip Administrator**

You are given a skeleton with a class **TripAdministrationsImpl** that implements the **TripAdministrations interface.**

The TripAdministrations works with **Company** & **Trip entities**, **all** **companies and trips** are identified by their **names and ids.** A company is allowed to have only **tripOrganizationLimit number of Trips to manage.** Implements all the operations from the **interface:**

**Note: All "get" methods return the elements in their order of addition.**

* **void addCompany(Company c)** – **adds** a company. If there **is** **a company with** **the** **same** **name** added before, throw IllegalArgumentException.
* **void addTrip(Company c, Trip t)** – **adds a trip for** the provided **company**. If the company does not exist, throw IllegalArgumentException.
* **boolean exist(Company c) –**  returns whether the **Company** has been **added** or **not.**
* **boolean exist(Trip t)** **–** returns whether the **Trip** has been **added** or **not.**
* **void removeCompany(Company c) –** **remove** the provided company with all its **trips**. If the company does not exist, throw IllegalArgumentException.
* **Collection<Company> getCompanies() –** return a collection of **all added companies**. If there are not any - return empty collection.
* **Collection<Trip> getTrips() – r**eturn a collection of **all added trips**. If there are not any - return empty collection.
* **void executeTrip(Company c, Trip t) –** **remove** the trip for the **provided company.** If the company or trip does not exist - throw IllegalArgumentException. If the trip is not for the provided company again - throw IllegalArgumentException.
* **Collection<Company> getCompaniesWithMoreThatNTrips(int n)** – return all companies with more than N trips.
* **Collection<Trip> getTripsWithTransportationType(Transportation t)** – **return all trips** filtered by the transportation type.
* **Collection<Trip> getAllTripsInPriceRange(int lo, int hi)** – return trips in between provided price range **inclusive**.

# **Trip Administrator – Performance**

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…

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…