Collections

Collection management - Additional exercises


Exercise 1:

Create a project called **ListBenchmark**. We are going to test in which cases is better to use an `ArrayList`, or a `LinkedList`. To measure the time an operation takes, you can use this piece of code:

```
Instant start = Instant.now();
// Some operation with ArrayList or LinkedList
Instant end = Instant.now();
Duration dur = Duration.between(start, end);
System.out.printf("ArrayList: The operation ... takes: %dms\n", dur.toMillis());
```

You have to compare these situations (`ArrayList<Double>` vs `LinkedList<Double>`). Create one of each empty and reuse the same lists in every comparison:

1. Add 100.000 (double) random items always at position 0. Compare times.
2. Delete the first 50.000 items (always delete the first one).
3. Add 50.000 random items in random positions.
4. Delete 50.000 items from random positions.

You'll see that when using a lot of random accesses (index), `ArrayList` is much faster (`LinkedList` needs to count from the beginning). When adding or deleting items at the beginning the situation is the opposite (`ArrayList` has to reorder internal indexes every time, whereas `LinkedList` doesn't need to).

Exercise 2:

Create a project named **Companies** with these classes (including `Main`):

- `Company` : Has name and money (double) attributes (values set in the constructor).
- `Person` : Has name and age attributes (values set in the constructor).
- In the main method, create a `TreeMap` (ordered by companies money) in which the key is a Company and the value is a `TreeSet` of `Persons` (ordered by person's age). Populate the TreeMap with 3 companies (do not add them ordered by money) and each company will have a list of 3 people (not ordered when adding). Iterate through the TreeMap (use the `.entrySet()` method to get a `Set` of keys ordered) and show the companies with their people (both should be ordered by money and age).

Exercise 3:

Create a project called **AnimalConversation** containing the following classes (including a `Main` class):

- An abstract class called `Animal`. It will have a protected attribute called `name` (string, assigned in the constructor), and an abstract method called `talk`, that will return a String with the sound that this animal produces.
- Classes `Dog`, `Cat` and `Sheep` that inherit from class `Animal` and implement the abstract method `talk` (the dogs will return "Wof wof", the cats will return "Meooow" and the sheeps will return "Beeee").
- A class called `AnimalConversation` that holds two objects (`animal1` and `animal2`), derived from `Animal`. Use generics to define the type of these two objects (each object can be from a different type but always derived from `Animal`). This class has a method called `chat()`, that prints in console the message from `animal1` (calling its method `talk()`), and then from `animal2`.
- In the `main` method, instance two or more `AnimalConversation` objects, each to compare different kind of animals (one can compare only `Dog` vs `Cat`, another `Cat` vs `Sheep`, and so on...). See that depending on how you define the generics in every instance it only allows you to set or get (implement setters and getters) that kind of animal and not others.