Object Oriented Programming

Exercise Series 5

In the following exercises, we will further investigate the potential of inheritance in Java. We will illustrate it by working on an architecture that will simulate a public transportation system. The idea is to have multiple means of transport (buses, subways, trains, ...) dealing with different types of passengers.

Exercise 1 First, design two classes that will set the basic requirements and communications a `Passenger` and a `Transport` must have:
A `Transport` must be able to

- return the number of seats currently available,
- return the number of standing places currently available,
- pick up a `Passenger`,
- notify its passengers it has reached the next stop.

A `Passenger` must be able to

- request a sitting/standing place in a given `Transport`,
- leave a `Transport` he entered after k stops,
- know his situation (inside a `Transport` or not, seated or standing, how many stops after he entered).

Consider that for some reason we do not want these classes to be directly instantiated, what options do we have in Java to forbid instantiation ? What are the implications of each option?
If we decide that the passenger capacity of a `Transport` should never change after instantiation, which solution seems the more adequate?

Exercise 2 Now, create a proper class of transportation `Bus` that can be instantiated. It should inherit from `Transport`.

Additionally to the usual features of a `Transport` (as described in ***exercise 1***), instances of the class `Bus` should also be able to count how many different `Passenger`s have travelled inside this `Bus` since it started its route.
What visibility would you give to the variables and other methods specific to the class `Bus`? What does that imply if we want to differentiate two kinds of buses in our program later on?

Exercise 3 Design the two following classes of `Passengers`:

- `SeatedPassenger` corresponding to passengers that will only enter a `Transport` seatted;
- `UncrowdedPassenger` corresponding to passengers that will only enter a `Transport` if it is not overcrowded (i.e. there is more that 10 standing places available when he enters).

Should the passengers know whether they are on a `Bus`, or simply that they are on a `Transport`? Could you have done ***exercise 3*** before ***exercise 2*** ? How does the *encapsulation* paradigm fits into this architecture?

—

Test your code with a short execution that uses all your implemented classes.

—

Exercise 4 The specification given in ***exercise 1*** of the interactions between the classes was not thorough. In particular, it did not specify precisely how the instances should behave in the event of a `Passenger` being denied access from a `Transport`.

First, take a look at your current code and check how this situation is handled, especially from the point of view of the `Passenger`.
Then, design a custom `Exception` class (called `TransportAccessException` for instance) and use it to convey a message to the `Passenger` to warn him that he did not get on the `Transport`.