Property Management

A property manager receives requests for intervention from the owners and dispatches them to suitable professional workers.
The main class is **`PropertyManager`**; all the classes are contained in the package **`managingProperties`**. The class **`Example`** provides usage examples for the main methods.

The JDK documentation can be found on a local server.

R1: Buildings, apartments and owners

The following methods allow the registration of the buildings in the system, together with the number of apartments and users.
The method **`addBuilding()`** accepts the (unique) id of the building (e.g. *`"b1"`*) and the number of its apartments, and records them. It throws an exception if the id has already been used or if it not in the range 1 to 100.
The method **`addOwner()`** accepts the (unique) id of the owner and the list of her apartments and record them. An apartment is identified by means of a string containing the id of the building and the number of the apartment, separated by ":" (e.g. *`"b1:10"`*). It throws an exception if the id of the owner has already been defined, the id of the building does not exist, the number does not correspond to an apartment, or the apartment already has an owner.
The method **`getBuildings()`** returns a sorted map that groups by number of apartments the lists of buildings sorted alphabetically.

R2: Professionals

The method **`addProfessionals()`** accepts the name of the profession (type of work) and the list of the ids of the relative professional workers. The professions used in the examples are electrician, plumber, mason (elettricista, idraulico, muratore). It throws an exception if the same profession has already been used in a previous invocation or if a worker's id has already been used: a worker can exercise a single profession only.
The method **`getProfessions()`** returns an ordered map of the professions (sorted alphabetically) and the corresponding number of workers.

R3: Maintenance requests

The method **`addRequest()`** accepts the id of the owner, the id of the apartment (e.g. *`"b1:10"`*), and the name of the profession; it generates a new request in the *`pending`* state and returns the number of the request. The requests are assigned a progressive number starting at 1. The method throws an exception if the owner, the apartment, or the profession do not exist, or if the owner does not own the apartment.
The method **`assign()`** assigns the request (whose number is provided as an argument) to the given professional worker and changes its status to *`assigned`*. It throws and exception if the worker does not exercise the profession required by the request, the request does not exits, or it is not in the pending state anymore.

The method **`getAssignedRequests()`** returns the list of the request numbers in ascending orders.

R4: Charges

To charge the expenses of a maintainance activity, the method **`charge()`** is used, it accepts the request number and the expenses sum (an integer number) and turns the request state into *`completed`*. It throws an exception if the requests does not exits, it is not in the assigned state, the sum is not in the 0 to 1000 range.
The method **`getCompletedRequests()`** returns the list of completed request numbers (i.e. in the completed state) in ascending order.

R5: Statistics

The methods **`getCharges()`** returns for each owner the sum of the relative completed requests. Only the owner with non null expenses are considered.
The method **`getChargesOfBuildings()`** returns, grouped by building, the sum of expenses by profession. Both buildings and professions are reported in alphabetic order. Only buildings and professions with non null expenses are considered