# Complementary exercise for Block 2

Create a project called **RealEstateAgency** in IntelliJ. We are going to implement an application to manage the data of several properties for sale. Then, define these packages in the project:

- `agency`
- `agency.property`

## Property class hierarchy

We are going to deal with some property subtypes. All these classes must be placed inside `agency.property` package.

There will be a parent, abstract class called `Property`, that will store the common information for all property subtypes: the property description (for instance "Small flat in the city centre"), the area of the property in m2, and the sale price (in euros, with no fraction digits). Define a constructor to set these three attributes, the getters/setters that you may need, and override the `toString` method to return the property data with this format: "Description (area, price)". For instance

```
Small flat in the city centre (50 m2, 90000 eur.)
```

There will be three Property subclasses:

- `Flat` subclass, which will add two new attributes to the ones inherited from `Property` superclass. These attributes are the number of rooms, and the floor number.
- `TerracedHouse` subclass, which will add the number of rooms and the terrace area.
- `Shop`, with one additional attribute to determine if it has a toilet or not.

Define appropriate constructors for these classes to re-use parent constructor when setting the common attributes. Also override the `toString` method of these subtypes to return the same information than parent's, with the additional attributes at the end of the string (with the format that you prefer). You can also add as many methods (including getters/setters) as you need.

Besides, each property will be assigned to an `Agent`, a person who is in charge of showing the property to every interested customer. Each agent will have a full name and a phone number, and each one can be in charge of many properties.

## The main class

Define a main class called `Agency` in agency package. It will have a main method to perform these steps:

1. It will create an array of up to 10 properties (`Property` objects, of any subtype).

2. It will ask the user to enter the information of each property, and then add it to the list. First of all, the user must choose the property type, and then ask for the attributes regarding this type. Regarding error control, we are going to assume that every data is correct, apart from the price: we need to check that it is a positive number, otherwise we will show an error message and ask for it again (until it is correct).
    - Regarding the agent for each property, there will be a pre-defined array of agents loaded in the application (create it manually in the code). Then, the user must choose one of these agents for each property. Keep in mind that a single agent can have many properties in his portfolio.

3. Step #2 will finish as soon as the user indicates that there are no more properties left to add (or when the properties array is full). You can choose how to detect this: either by an empty string at the beginning of each property, or with some special string input (such as "FINISH"), etc.
4. Once the properties have been added to the list, there will be a loop showing a menu with these options:
    1. **Search by type**: the user will choose the property type (flat, terraced house or shop) and then a list of all the properties of this type will be shown.
    2. **Sort by price**: it will show a list with all the properties sorted by price in ascending order. You must use a `Comparator` interface to sort the properties, either with a new class, or with an anonymous class.
    3. **Search by area**: the user will enter an area in m2, and the program will show a list with all the properties whose area is greater or equal than the specified area.

    - In all these options, data must be shown according to the `toString` method of every object, and it must include the information about the Agent in charge of each property.

    4. **Exit**: to exit the program

## Class diagram

You must also design the class diagram to represent this project, including classes (with attributes, constructors and methods) and relationships.

## Evaluation criteria

- Class diagram of the project: 1 point
- Project and packages properly defined: 0,5 points
- Property parent, abstract class with appropriate attributes, constructor and method(s): 1 point
- Property subclasses, with appropriate constructors, method overriden, etc: 1,5 points
- Agent class with its attributes, constructor and other methods, and an appropriate relationship with the properties: 1 point
- Initialize arrays from main program and ask the user to enter properties: 2 points
- Search by type menu option properly implemented: 0,5 points
- Sort by price menu option properly implemented with the appropriate interface implementation: 1 point
- Search by area menu option properly implemented: 0,5 points
- Code cleanliness and reusability: 1 point

> **VERY IMPORTANT**: every exercise that does not compile will be automatically evaluated to 0.