Block 2 - Exercise for Units 5 to 7

We want to implement a home automation system using object oriented programming. To do this, we are going to create an IntelliJ project called **HomeAutomation**. Inside the source folder, you must define two packages:

- `automation.main` : inside this package we will place the main class of the system, called `Main` , and an additional class called `Management` .
- `automation.data` : inside this package we will add all the classes and interfaces needed to deal with the automation system.

1. Automation elements

We are going to take into account the following elements, that need to be included as separate classes/source files inside `automation.data` package.

Blinds

We will define a class called `Blind` . Every blind must have the following methods

- `raise()` : it will completely raise the blind
- `lower()` : it will completely lower the blind
- `raise(int percent)` : it will raise the blind the specified percent. If a blind is raised a 40%, for instance, and we call `raise(20)` , then the blind will be raised a 60%. You must make sure that percent is a correct value before raising the blind.
- `lower(int percent)` : it will lower the blind the specified percent, just as we do to raise it.
- `getPercent()` : it will return the current level (percentage) of the blind

Besides, we need to define a constructor to specify the initial blind level (percentage: 0 = completely down, 100 = completely up)

Windows

We will also define a class called `Window` . Every window will have a blind, so that we need to set the blind associated to the window in the constructor. Besides, there will be a *getter* to get the blind associated to this window. Finally, windows can be locked or unlocked through the corresponding methods `lock()` and `unlock()` . We can also check the status of the window (locked or unlocked) with `getStatus()` method.

Awnings

We are going to consider that awnings are just subtypes of blinds. So we need to define a class called `Awning` that will inherit from `Blind` class.

Doors

There will be a generic `Door` class, with the methods `lock()` and `unlock()`, along with a `getStatus()` method to check if it is currently locked or unlocked. The constructor of this class will establish if the door is initially locked or unlocked.

We also need a special subtype of door, which is the `GarageDoor`. This class needs to have the same methods as `Blind` class: `raise()`, `raise(percent)`, `lower()`, `lower(percent)` and `getPercent()`.

Heating

Define a `Heating` class, in which we can specify the temperature (in ºC), along with switching on and off the system through methods `switchOn()` and `switchOff()`. There will be a *getter* and *setter* to get/set the current temperature, and a `getStatus()` method to get heating's current status (switched on or off). Besides, we need two constructors:

- One that will receive just the initial temperature, and then the heating will start switched off.
- Another one that will receive both elements (temperature and initial status).

We need to define a subtype of heating called `Oven`, with the same elements as `Heating` class.

Lights

Finally, we need a `Light` class. Every light can be switched on and off, and we can also check its current status.

General information

Every automation element will have a name (for instance "Kitchen window", or "Heating for Room 1"), a *setter* to set this name, and a `toString` method to return the relevant information about this element (for instance, the name, temperature and current status in case of a heating system, or the name and current position of an awning).

You may need to define a generic, abstract class called `AutomationElement` from which every automation element (blinds, windows, doors) will inherit (directly or indirectly).

Shared information

Notice that many classes in our system share a common behavior: blinds and garage doors can raise or lower, either completely or a given percentage. Windows and doors can be locked or unlocked. Also, heatings and ligths can be switched on and off. You must include the necessary elements (interfaces) to properly share this behavior among these classes.

Additional source code

You can add as many additional elements (classes, interfaces and/or methods) to meet the requirements of the exercise, but you MUST include the specified ones, with their appropriate names and content.

2. The *main* package and application

Inside `automation.main` package there should be two classes:

Management

This class will be in charge of managing the different components of a house. There will be an `AutomationElement` array to store all the elements in the house. In the constructor, we must initialize it with 2 windows, one awning, a main door, a garage door, 2 lights, 2 heating systems and one oven, with these names:

- Windows: "Room window" and "Living room window"
- Awning: "Terrace awning"
- Main door: "Main door"
- Garage door: "Garage door"
- Lights: "Kitchen light" and "Room light"
- Heatings: "Room heating" and "Bathroom heating"
- Oven: "Kitchen oven"

This class must provide methods to set up the automation elements (switch on/off the lights, raise or lower blinds, and so on) according to main's menu, explained below.

Main

Main class will initialize the `Management` object with default values (nothing must be asked to the user in this stage). Then, it will show this menu to the user:

- **1. Winter mode**: with this option, every door and window will be locked, garage door will be completely lowered, every blind will be set to 80% and terrace awning will be set to 100% (completely raised). Besides, every heating will be switched on to 25ºC, and every light will be switched on.
- **2. Summer mode**: with this option, every door and window will be unlocked, garage door will be completely raised and every blind and awning will be set to 50%. Finally, every heating, oven and light will be switched off.
- **3. Cooking mode**: with this option, kitchen light and oven will be switched on, and room light will be switched off. The oven will be set to 200ºC.
- **4. Close everything**: this step will consist in:
    - Locking every door
    - Lowering every garage door, awning or blind (completely)
    - Switching off every heating, oven or light
- **5. Show status**: an overall status of the whole automation system will be shown, indicating the relevant information of every component (`toString` methods can be really helpful in this step). This is an example of what this should show:

```
Overall status of the house:
- Room window: 20%
- Living room window: 40%
- Terrace awning: 0%
- Main door: locked
- Garage door: locked, 0%
- Kitchen light: switched off
- Room light: switched on
- Room heating: switched on, 25ºC
- Bathroom heating: switched off, 24ºC
- Kitchen oven: switched off, 200ºC
```

- **0. Exit**: to exit the application

We assume that user will always type a valid menu option.

3. Class diagram

You must also include a class diagram (in either JPG, PNG or PDF format) for the whole system specification. You can use Visual Paradigm Online, Modelio or any other software tool to do it.

4. What to submit?

You must submit a ZIP or RAR file containing:

- The whole IntelliJ project with all the system implemented
- A JPG, PNG or PDF file with the class diagram

5. Evaluation criteria

This exercise will be evaluated as follows:

- Class diagram: 2 points
- Correct project definition, including package structure and class names: 0,5 points
- `AutomationElement` parent, abstract class: 0,5 points
- `Blind` and `Awning` classes: 0,5 points
- `Window` class: 0,5 points
- `Door` and `GarageDoor` classes: 0,5 points
- `Heating` and `Oven` classes: 0,5 points
- `Light` class: 0,5 points
- Additional elements (interfaces) needed to share behavior properly between classes: 0,5 points
- Code reuse, or no code repeated unnecessarily: 0,5 points

- `Management` class, including all the automation elements needed and the methods to set up each element: 2 points
- `Main` class to interact with user and management system with the corresponding menu options: 1 point
- Code cleanliness: 0,5

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