Block 2 - Exercise for Units 5-6

We want to implement an application to manage the user reviews for some business using object oriented programming. To do this, we are going to create an IntelliJ project called **BusinessReviews**. Inside the source folder, you must define two packages:

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

1. Application data

The following classes must be placed inside `reviews.data` package.

1.1. Business types

We are going to define a parent, abstract class called **Business**. From each business, we want to store its name and location (city name). Define a parameterized constructor, along with the corresponding *getters* and *setters*, and override the *toString* method to return the business name with the location between parentheses. For instance:

```
Giovanni Pizzeria (London)
```

From this parent class, we are going to define three different subtypes:

- Restaurants (in a class called **Restaurant**). In this case we want to store the food type (italian, mediterranean...) as an additional attribute. Define the constructor and the appropriate *getter/setter* and override *toString* method to specify that this business is a restaurant, along with the food type. For instance:

```
Restaurant Giovanni Pizzeria (London) - italian
```

- Hairdressers (in a class called **Hairdresser**). We need to know if this hairdresser is unisex or not (a boolean value). Define the corresponding constructor and *getter/setter*, and override *toString* method to specify that this business is a hairdresser, indicating whether it's unisex or not. For instance:

```
Hairdresser John & Mary (Edinburgh) - unisex
```

- Garages (in a class called **Garage**). We add the price per hour of the workforce in euros (floating point number) as its particular attribute, along with the constructor and *getter/setter*. The *toString* method must be overriden to show that this business is a garage, along with the price per hour. For instance:

```
Garage Road 66 (Liverpool) - 30 eur/h
```

These subclasses must rely on parent class as much as possible (constructors, overriden methods and so on).

1.2. Users

There will be a **User** class to store registered users information: only *login* and *password* are needed, so define the corresponding constructor and *getters* (not *setters* nor *toString* method needed here).

1.3. Reviews

Finally, there will be a **Review** class to store the different reviews. Each review must have an associated user (one user per review), along with the comment (text) and the rating (integer between 0 and 5, both included). Define the constructor to specify these three values (user object, comment and rating), the corresponding *getters/setters* and a *toString* method that prints the information of the review this way:

```
User login
Comment
Rating
```

For instance:

```
john123
It's a fantastic restaurant. The food is really good and the staff is very kind.
5/5
```

1.4. Association between businesses and reviews

Every business will have an array of reviews as its internal attribute. Add it to the parent abstract class, and assign it in the constructor, along with a *getter* to get this array from the business. Besides, add a new method to *Business* class called `reviewAverage`, with no parameters; it will return the average of the reviews belonging to this business

The *toString* method of every business must now show this additional information. For instance:

```
Restaurant Giovanni Pizzeria (London) - italian
   Review average: 4.3
```

2. The main package

Regarding the main program, its code will be divided in two source files: **Main** class and **Management** class.

2.1. The Management class (part I)

We are going to define most of the code inside **Management** class (in `reviews.main` package). As internal attributes of this class, we need to add an array of users and an array of businesses. Then, you need to add the following methods:

- `void initialize()`: with no parameters nor return type. This method will be in charge of:
  - Initialize the users array with at least 10 different users (with their corresponding logins and passwords)
  - Initialize the business array with at least 2 businesses of each type. For each business, we need to define an array of at least 2 reviews, coming from different users of the previous array. There can't be more than one review from the same user in the same business.

> **NOTE**: this method does NOT need to ask anything to the user. Every object must be defined manually in the code.

- `User login(string login, string password)`: it will search a user with the corresponding login and password in the user array, and return it. If no user matches the credentials, it will return *null*.

2.2. The Main class

Inside the *Main* class we just need a `main` method. It must complete these steps

- Instantiate a *Management* object and call its *initialize* method
- Ask the user to login (repeatedly until we get a valid *User* object). You must use *login* method from *Management* object.
- Show a menu to the user with these options:
  1. *My reviews*. It will show all the reviews belonging to current user (from all the businesses stored in *Management* class)
  2. *Business list*. It will show all the business from the *Management* class sorted by name alphabetically
  3. *Top rated businesses*. It will ask the user to enter a business type (for instance, 1 for restaurants, 2 for hairdressers and 3 for garages) and then show the 3 most relevant businesses sorted by rating average in descending order.
  4. *Edit my review*. It will ask the user to choose a business, and if the user has a review available for this business, then he/she can change the comment and/or rating for this review in this business.

5. *Quit*

2.3. The Management class (part II)

In order to meet the requirements of previous menu, you may need to add some additional methods to *Management* class. For instance:

- A method `void showReviews(User user)` to show the reviews belonging to a given user. It can receive as parameter the user object (or user login) and then explore the business array and print the reviews of this user
  - This method can be used from option 1 of previous menu
- A method `void sortBusinessesByName()` to sort the businesses alphabetically by name and show them
  - This method can be used from option 2 of previous menu
- A method `void sortBusinessesByRating(int type)` to sort the businesses by rating average in descending order, and show the first 3 businesses of a given type.
  - This method can be used from option 3 of previous menu
- A method (or methods) to help the user edit a review (option 4 of previous menu). For instance:
  - A method `public Business findBusiness(String name)` to search a business by its name in the array of businesses, and return it (or *null* if it has not been found)
  - Another method `public Review findReview(User user, Business business)` to find a review of a given user in a given business and return it (or *null* if it has not been found)
  - Another *static* method `changeReview(Review r, String comment, int rating)` that updates the comment and rating of the review *r* with the new information stored in *comment* and *rating* parameters. So, from the main function, you can ask the user to type the new comment and rating, and then call this method to update it.
  - Note that you must make sure that data exists before trying to use it. For instance, if user types a business name that does not exist in the business array, then we can't go on in this option, and we must return to menu.

3. Final considerations

Apart from the classes and methods mentioned above, you are free to add as many additional methods to each class as you need (*Main* class must ONLY have the *main* method). Also, you can add as many interface implementations (either using classes or anonymous classes) as you need, specially in order to sort business by different criteria.

You must deliver **the whole *IntelliJ* project compressed in a single ZIP/RAR file**.

4. Evaluation criteria

0. Project structure with the appropriate package names and classes inside each package: **0,25 points**
1. Application data
   - Parent, abstract *Business* class, including association with *Review* class: **1 point**
   - Business subclasses, including code reusability: **0,5 points**
   - *User* class: **0,25 points**
   - *Review* class, including association with *User*: **1 point**
2. Main package
   - *Management*'s *initialize* method: **0,5 points**
   - *Management*'s *login* method: **1 point**
   - Management instantiation and user login from *main*: **1 point**
   - *My reviews* option (including appropriate method definition): **1 point**
   - *Business list* option (including appropriate sorting and method definition): **1 point**
   - *Top rated businesses* option (including appropriate sorting and method definition): **1 point**
   - *Edit my review* option (including appropriate method decomposition, and using a *static* method in this section): **1 point**
3. Code cleanliness and comments in every source file: **0,5 points**