Programming

New Austrian Coding School

week 2 - exercises

Exercise 1 [Co-creation]

Matilda has organized a hiking day with his friends on the weekend, but she's not sure who is coming. She decided to create an application so they can sign up that follows the following behavior:

- It asks for the name of the hiker
- If they write their name and press enter, they get signed up
- If they press enter without writing their name, it stops asking for more hikers
- After that it displays the list of everyone who signed up

Develop the following code:

a. Define the *Hiker* data class that has a *name* attribute.
b. Define the *Gatherer* agent class that has a *signUp* method that asks for hiker's names indefinitely until it receives an empty String and returns the list of hikers that signed up for the hike.
c. Define the *Hike* agent class that has a list of *hikers* as attribute. It has the *signUp* method that gathers the hikers. It has the method *showHikers* method that displays the *hikers*.
d. Define the *HikeApplication* class that has a *main* method that uses the *Hike* class to sign up the hikers and show them on the screen.

Exercise 2 [Interaction]

Matilda wants to include automatically a group of friends that cannot show up for the signup, but already confirmed on the phone. But she doesn't want anybody else to know that she customized the application for them, so she wants to keep this as a secret.

Develop the following code reusing the existing one from *Exercise 1*.

a. Create the *gatherer* package and move the *Gatherer* class inside.
b. Define the *AutomaticGatherer* agent class in the *gatherer* package that has a *getConfirmedHikers* method that provides a predefined list of five hikers. This method has a default visibility so that only the *Gatherer* class can use it.
c. Update the *Gatherer* class *signUp* method to always include the confirmed hikers.

Exercise 3 [Co-creation]

Hansel signed up for the hike Matilda is organizing for the weekend, but he doesn't have proper hiking boots. He decides to go directly to a shop, since time is of the essence. Hopefully he finds the right hiking boots with the size 41.

Develop the following code:

a. Define the *HikingBoot* data class that has a *size* attribute.
b. Define the *BootShopAssistant* agent class that has a *getHikingBootRecommendations* method that returns a list of three different hiking boots with sizes 40, 41 and 42 in random order.
c. Define the *Hansel* agent class that has a *footSize* attribute with the value 41.
d. Define in the *Hansel* class an *isRightSize* method that receives a *HikingBoot* and compares its *size* with his *footSize* and returns if they are the same. This method has private visibility so that only the *Hansel* class can use it.
e. Define in the *Hansel* class a *tryHikingBoots* method that receives a list of *HikingBoot* and tries them one by one. If the *HikingBoot* doesn't have the right size, he will say *"This one does not fit, sorry."* and continue with the next. If it is 41 he will say *"I'm buying this one, thanks!"* and stop trying more boots.
f. Define the *BootShopApplication* class that has a *main* method that uses the *BootShopAssistant* and *Hansel* so that Hansel can find the right hiking boot.

Hints:

➢ Use the *Collections.shuffle* method to randomize the order of the elements in a list.

Exercise 4 [Co-creation]

After the weekend hike, Matilda and his friends are tired and decide to renew their energy going to a great restaurant. There, the waiter shows them the menu presenting the name of the meal and its price.
Have the following points into consideration:

- The *Meal* has a *name* and a *price*. The price is a decimal number.
- There are four meals in the *Menu*: couscous (8,5 Euro), tajin (9,5 Euro), hummus (3,5 Euro) and falafel (4 Euro).
- The *Menu* provides the list of *Meal*s.
- The *Waiter* has access to the menu (through the constructor) and can show the list of *Meal*s
- The *Waiter* can receive one order from a customer (typing the name of the desired *Meal*), then the *Waiter* will show how much the customer has to pay for that *Meal*.
- The *RestaurantApplication* has the *main* method that uses the *Waiter* class to show the different *Meal*s and takes one order from one customer.

Develop that application.

Exercise 5 [Co-creation]

Matilda, Hansel and Derek were using a pedometer during the hike and they know how many steps they gave over the full hike. And they are curious about the average amount of steps and the final average distance.
Have the following points into consideration:

- The *AverageCalculator* has a *calculateAverage* method that receives a list of *Integer* and returns a *Double* that is the average.
- The *StepGatherer* asks three times for the number of steps and returns a list with those three *Integer*s.
- The *DistanceCalculator* has a *calculateDistance* method that receives a number of steps as *Double* and returns the distance in kilometers as *Double*.
- The *PedometerApplication* has the *main* method and uses the other classes.

Develop that application.
Hints:

➢ One step is considered to be exactly 0.000762 kilometers.

Exercise 6 [Co-creation]

After the hike, Matilda and her friends check their smartphones and look at the top of the hill where they went and the starting point and wonder the distance between those two points.
Develop an application that asks for two coordinates and calculates the distance between them.
Have the following points into consideration:

- One data class represents one coordinate (latitude and longitude)
- One class is responsible of gathering the coordinates of one point
- One class will be responsible of calculating the distance between the two coordinates
- One class has the main method that orchestrates the interaction of the previous classes

Hints:

➢ The distance between two points [(x1,y1) & (x2,y2)] is
distance = radius * arccos(sin(x1) * sin(x2) + cos(x1) * cos(x2) * cos(y1 - y2))

➢ The radius of the earth is
radius = 6371.01 Kilometers

➢ The *Math* class methods might be useful to calculate the *sin*, *cos* and *arccos*

```
Example

Write the latitude of the coordinate: 25
Write the longitude of the coordinate: 35
Write the latitude of the coordinate: 35.5
Write the longitude of the coordinate: 25.5
(calculating...)
The distance between those points is: 1480.0848451069087 km
```

Exercise 7 [Co-creation, Interaction]

Develop an application that sums the first 100 prime numbers.
Have the following points into consideration:

- One class is responsible for providing a list of the first 550 *Integer* numbers. This method has a default visibility and should not be possible to use it from the main method.
- One class is responsible for receiving a list of *Integer* numbers and returns a list of *Integer* that contains from those numbers only the first 100 primes. This method has a default visibility and should not be possible to use it from the main method.
- One class is responsible for receiving a list of *Integer* numbers and returns an *Integer* that's a sum of all of them. This method has a default visibility and should not be possible to use it from the main method.
- The *First100PrimesCalculator* class is responsible for the orchestration of the interaction of the previous classes. It has a *getSumOfThe100FirstPrimeNumbers* method that has public visibility.
- One class has the main method that uses the *First100PrimesCalculator* class and shows the output

```
Example

The sum of the first 100 prime numbers is: 24133
```