Programming

New Austrian Coding School

week 3 - exercises

Exercise 1

The railway company wants to give a special treatment to their business class customers. On the top of each train seat there is a small screen displaying the name of the customer. They want those names of the business class to be displayed in capital letters and those in economic class to be displayed in small letters.
Develop the code that the screen uses to display the customer name.
Follow the instructions below:

- You need the following classes: *Customer, Screen* and *ScreenApplication.*
- A *Customer* has a *name* and a *category*. It also has the methods *getName* and *getCategory*.
- A *Screen* can *display* a customer *name* according to the railway rules*,* and it also returns the *name* it displayed so that it can be tested afterwards.
- In the *ScreenApplication* class *main* method, create a business class *Customer*, an economic class *Customer,* a *Screen* and *display* them the *screen*.
- [Testing] Create the *ScreenTest* class and test its behavior. Write three methods: *testBusinessCustomer, testEconomicCustomer* and *testOtherCustomer*. What happens when a business class has to be displayed? What happens when the customer is in economic class? What if they have a different category or none at all?

Hints:

➢ The *equals, toUpperCase, toLowerCase* from the String class might be useful.

Exercise 2

An insurance company wants to send customized emails to their customers. If the *name* starts with *Mr.* they want to send the *man* template. If it starts with *Ms.* or *Mrs.* they want to send the *woman* template. Also, the *Hokopoko* family is the owner of the company, and if the email would be sent to anyone of that family, they want to send the *privileged* template. If neither of the previous cases are recognized through the name, then they want to use the *default* template.
Develop the code that the insurance company uses in their email application to choose the right template.
Follow the instructions below:

- You need the following classes: *Customer, TemplateChooser* and *InsuranceApplication*.
- Reuse the *Customer* from exercise 1.
- The *TemplateChooser* has no attributes. It has the *chooseTemplate* method that receives a *customer* and returns the word *man, woman, privileged* or *default* depending on the insurance company's requirements.
- In the *InsuranceApplication* class *main* method, create several *Customer*s and a *TemplateChooser* and see the results.
- [Testing] Create the *TemplateChooserTest* class and test its behavior. Write four methods: *testManTemplate, testWomanTemplate, testPrivilegedTemplate* and *testDefaultTemplate*. What happens in each one of the cases? What happens when there is a space at the beginning of their name?

Hints:

➢ The *startsWith, contains* and *trim* methods from the String class might be useful.

Exercise 3

Some kids have been messing around with the weight machine of the supermarket. The numbers displayed on the screen to select the good you want to weigh are sometimes appearing as negative or decimal numbers. Customers are confused because they usually pressed on numbers starting from 1 and they were never decimal.
Develop the code that the supermarket uses to fix the bug that the kids found.
Follow the instructions below:

- You need the following classes: *NumberCorrector,* and *SupermarketApplication*.
- The *NumberCorrector* class has no attributes. It has the *correct* method that receives a Double and returns the Integer version of it in absolute terms.

- In the *SupermarketApplication* class *main* method, create one Double number and a *NumberCorrector* and correct it to see the results.
- [Testing] Create the *NumberCorrectorTest* class and test its behavior. Write the method *testCorrect*.

Hints:

➢ The *Math.floor(double)* and *Math.abs(number)* methods might be useful.

Exercise 4

A network of hackers is communicating between each others with an uncommon messaging system. They agreed on the following points:

- They will never write numbers as numbers in their messages. Instead they will write the word for the numbers. So if they want to write *5*, they will write *five* instead.
- They will never use mathematical symbols. Instead they will also write the proper names of those symbols.
- They are allowed to use the punctuation symbols *comma* and *dot.* The rest are not allowed.
- They will only use small letters
- When they write a message, the vowels will be replaced so:
  - *a* become a 4.
  - *e* become a 3.
  - *i* become a 1.
  - *o* become a 8.
  - *u* become a 9.
- After the replacement has taken place, the message will be encrypted by adding a random number (from 1 to 5) of rubbish symbols between all letters. The rubbish symbols can be chosen among: *!@#$%&*+-=*
- Then the message is sent through a super secure internet tunnel.
- When the message is received, the process of encryption is reversed removing the rubbish symbols and rearranging the vowels.

Develop the code that the hacker network uses to encrypt and decrypt their messages.
Follow the instructions below:

- You need the following classes: *Encryptor, Decryptor, Messenger* and *HackerApplication*.
- The *Encryptor* class has no attributes. It has the *encrypt* method that receives a String and returns the encrypted version of it.
- The *Decryptor* has no attributes. It has the *decrypt* method that receives a String and returns the decrypted version of it.
- The *Messenger* has an *encryptor* and a *decryptor* as attributes. It has the *send* method that receives a String, encrypts it and displays the encrypted message. It has the *receive* method that receives a String, decrypts it and displays the decrypted message.
- In the *HackerApplication* class *main* method, create one String message and a *Messenger* and send and receive it to see the results.
- [Testing] Create the *EncryptorTest* class and test its behavior. Write the method *testEncrypt*. What happens if the message already contained numbers in the form of 4 or 5? Could this be somehow solved to help hackers who forget about this rule?
- [Testing] Create the *DecryptorTest* class and test its behavior. Write the method *testDecrypt*.

Hints:

➢ The *replaceAll* and *split* methods of the String class might be useful.
➢ If you use *split* without a delimiter, it splits every letter of a string into an array of string.
➢ The *nextInt(limit)* method of the Random class might be useful.
➢ The Collections.rotate(collection, shift) method might be useful.
➢ It's up to you how much you want to randomize the rubbish characters *!@#$%&*+-=*. You could choose one per letter and rotate the list, or you could choose one character at random per letter. Once you have chosen the right one, you have to repeat it a random number of times between 1 and 4.