Programming

week 9 - exercises

New Austrian Coding School

Exercise 1

Develop the following code:

1. Create the *StreamsApplication* class. Write the next points of this exercise inside the *main* method.
2. Create a list of the first twenty *numbers*. Use a loop for this, please.
3. Create a list of the even *numbers.* Display it.
4. Create a list of the odd *numbers.* Display it.
5. Create a list of the *numbers* divisible by 3 and bigger than 10. Display it.
6. Create a list of the *numbers* smaller than 5 and multiply them by 3. Display it.
7. Create a list of String where every element follows the format *"number <number> has <digit> digits"*. The numbers considered have to be bigger than 8 and smaller than 12. Display it.

```
Even numbers:
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
Odd numbers:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
Numbers divisible by 3 and bigger than 10:
[12, 15, 18]
Numbers smaller than 5 and multiplied by 3:
[3, 6, 9, 12]
Digits of numbers bigger than 8 and smaller than 12:
[number 9 has 1 digits, number 10 has 2 digits, number 11 has 2 digits]
```

Exercise 2

Develop the following code:

1. Create the *StreamsTest* class. Write the next points of this exercise as tests asserting the expectations.
2. Use the *FileReader* to create an attribute called *names* that is a list of String reading the *names.txt* file.
3. Create a list with the *names* that are shorter than four characters. Check that the output is correct.
4. Create a list with the *names* that end with the letter *"m"* and make them upper case. Check that the output is correct.
5. Create a list with the *names* that contain the letters *"e"* and *"r"*. Check that the output is correct.
6. Create a list with the *names* that are exactly four characters long, make them lower case, keep those that contain the letters *"a"* and *"m"* and duplicate the number of letter *"a"*s that they have.

Notes:

- Use the *FileReader* provided in the *reader* package. Note that it has the *asStream* method, because files can be read with Streams as well.
- Use the *names.txt* file provided in the source package.

```
Names that are shorter than four characters:
[El]
Names that end with m uppercased:
[HOSAM, TAMMAM]
Names that contain the letter e and the letter r:
[Mehran, Norbert, Serife]
Names that are 4 letters long in lowercase and contain the letter a and m with double a's:
[maarj, aamin, omaar]
```

Exercise 3

Develop the *Caesar* cipher exercise:

1. Create the *Caesar* class. It has the *cipher* method that receives the *plaintext* and the *key* and returns the *ciphertext*.
2. Create the *CaesarTest* class and provide meaningful tests.

Rules:

- You're not allowed to use a loop.
- The first line of the *cipher* method starts with the *return* command.
- You're allowed to create other methods

Hints:

- ➢ *You can refresh your memory about the Caesar description here.*
- ➢ *You can start a Stream using the Stream.of(...) method.*
- ➢ *You can use one method that you have created inside a Lambda.*

Exercise 4

Winter has arrived. Now that we are surrounded by cold and silence we start thinking about stuff in general, like: is the country where I live one of the bests where I could be? Fortunately we can already answer these questions with our programming skills!
Develop the code that displays on the screen the five best countries to live.
Follow the points below:

- Use the *world-happiness-2017.csv* file provided in the source package.
- Use the *FileReader* provided in the *reader* package.
- Create the *HappinessRecord* class that has a *country*, a *rank* and a *score*.
- Create the *HappinessApplication class.*

Rules:

- You're not allowed to use a loop.
- You're only allowed to write one semicolon (;) in the *main* method. Using *";"* in the *split* method doesn't count.

Hints:

- ➢ *You can skip the header of the file by using the "skip(number)" intermediate operation.*
- ➢ *You can limit the number of elements of the Stream by using the "limit(number)" intermediate operation.*
- ➢ *You can sort the elements of the Stream by using the ".sorted(Comparator.comparing(e -> e.getRank()))" intermediate operation.*

```
Rank: 1 | Country: Norway | Score: 7.53700017929077
Rank: 2 | Country: Denmark | Score: 7.52199983596802
Rank: 3 | Country: Iceland | Score: 7.50400018692017
Rank: 4 | Country: Switzerland | Score: 7.49399995803833
Rank: 5 | Country: Finland | Score: 7.4689998626709
```

Exercise 5

We have a film database with more than 5.000 films. Let's see what curiosities we can find out from the film industry!
Develop the following code:

- Use the *films.csv* file provided in the source package.
- Use the *FileReader* provided in the *reader* package.
- Create the *Film* class that has a *title*, a *score*, a *vote count,* a *runtime*, a *budget* and a *revenue*.
- Create the *FilmReader* class. It has the method *getFilms* that returns a list with all the films in *films.csv*.
- Create the *FilmSummarizer class.* Each of the following questions will be answered with a method in this class.

Questions:

- Which are the three films with the highest rating, sorted by rating?
- Which are the three films with the highest rating that are longer than three hours, sorted by rating?

- Which are the four most expensive films, sorted by budget?
- Which are the four most expensive films that are shorter than one hour and a half, sorted by budget?
- Which are the four most rated films with a rating higher than 7 and a budget between 50.000 and 500.000 dollars, sorted by budget?
- Ask two additional questions that you might find interesting.

Rules:

- You're not allowed to use a loop.
- Every method of the *FilmSummarizer* class starts with the *return* keyword.

Hints:

- ➢ *You can avoid reading the movies.csv file constantly in the FilmSummarizer if you have them as an attribute.*
- ➢ *You can skip the header of the file by using the "skip(number)" intermediate operation.*
- ➢ *You can limit the number of elements of the Stream by using the "limit(number)" intermediate operation.*
- ➢ *You can sort the elements of the Stream by using the ".sorted(Comparator.comparing(Film::getScore).reversed())" intermediate operation.*

```
Top 3 with highest rating, sorted by rating:
["Stiff Upper Lips", "Dancer, Texas Pop. 81", "Me You and Five Bucks"]

Top 3 with highest rating longer than 3 hours, sorted by rating:
["Schindler's List", "The Godfather: Part II", "The Green Mile"]

Top 4 most expensive, sorted by budget:
["Pirates of the Caribbean: On Stranger Tides", "Pirates of the Caribbean: At World's End",
"Avengers: Age of Ultron", "Superman Returns"]

Top 4 most expensive shorter than 90 minutes, sorted by budget:
["Madagascar: Escape 2 Africa", "G-Force", "Mars Needs Moms", "Chicken Little"]

Top 4 most rated over 7 and budget between 50.000 and 500.000, sorted by budget:
["Monty Python and the Holy Grail", "Intolerance", "The Evil Dead", "12 Angry Men"]
```