Programming

New Austrian Coding School

week 4 - exercises

Exercise 1 [data structures - lists]

Derek is checking his fridge thinking about what to buy at the grocery store. He decides to write down a shopping list.
Create the *BasicShoppingApplication* class and write the following instructions in its *main* method:

- Create an empty *groceries* list and display it.
- Create three Strings that represent three different groceries.
- Add them to the *groceries* list and display it.
- Remove one of them by providing its name as an argument and display the list.
- Add one of the existing groceries twice and display the list.
- Remove the third element of the list by providing its position and display it.
- Display its size.

```
Groceries: []
Groceries: [bananas, oranges, tomatoes]
Groceries: [bananas, tomatoes]
Groceries: [bananas, tomatoes, tomatoes, tomatoes]
Groceries: [bananas, tomatoes, tomatoes]
Groceries size: 3
```

Exercise 2 [data structures - sets]

Hansel is preparing himself for a trip and wants to make sure he has packed at least one of everything.
Create the *BasicBackpackApplication* class and write the following instructions in its *main* method:

- Create an empty *backpack* set and display it.
- Create three Strings that represent three different items.
- Add them to the *backpack* set and display it.
- Ask if one of them is contained in the set and display the answer.
- Remove one of them by providing its name as an argument and display the set.
- Ask if the element you just removed is not contained in the set and display the answer.
- Add one of the existing items twice and display the set.
- Display its size.

```
Backpack: []
Backpack: [toothpaste, towel, underwear]
The backpack contains underwear
Backpack: [toothpaste, towel]
The backpack no longer contains underwear
Backpack: [toothpaste, towel]
Backpack: [toothpaste, towel]
Backpack size: 2
```

Exercise 3 [data structures - maps]

Matilda just bought a new book about social impact and as she opens it, she sees the index
Create the *BasicIndexApplication* class and write the following instructions in its *main* method:

- Create an empty *index* map connecting Integers for the page numbers and Strings for the book topics and display it.
- Create three Integers that represent three different page numbers.
- Create three Strings that represent three different book topics.
- Put them into the *index* map from biggest to smallest page number and display it to notice that the entries are automatically arranged.

- Display the map keys.
- Display the map values.
- Create one String representing another book topic.
- Put this new topic into the same page number than an existing one and display the map to notice that the previous one is replaced.
- Ask if one of the keys is contained in the map and display the answer.
- Get one of them by providing its name as an argument and display it.
- Display its size.

```
Index: {}
Index: {1=connecting, 2=possitive things, 3=ecologic economy}
Index keys: [1, 2, 3]
Index values: [connecting, possitive things, ecologic economy]
Index entry sets: [1=connecting, 2=possitive things, 3=ecologic economy]
Index: {1=teaching methods, 2=possitive things, 3=ecologic economy}
The page number 1 exists as key
The topic in page number 3 is ecologic economy
Index size: 3
```

Exercise 4 [optionals]

Christmas is coming and Derek's new working colleges want to organize an "invisible friend" present giving in their company.
Create the *BasicGiftApplication* class and write the following instructions in its *main* method:

- Create an empty *giftBox* Optional of String and display it.
- Assign an Optional of String to the *giftBox* containing an actual present and display it.
- Ask if the *giftBox* is present and display the answer.
- Get the value contained in *giftBox* and display it.

```
Gift: Optional.empty
Gift: Optional[Robotoy]
There is a present in the gift box
The gift is a Robotoy
```

Exercise 5 [data structures - lists]

Derek is checking his fridge thinking about what to buy at the grocery store. He decides to write down a shopping list.
Create the *AdvancedShoppingApplication* class and write the following instructions in its *main* method:

- Create the *Grocery* class that has one *name*.
- Create an empty *groceries* list of *Grocery* and display it.
- Create three different *Grocery* objects.
- Add them to the *groceries* list and display it. To display the *Grocery* class create the *toString* method.
- Create a new *Grocery* with the same *name* as a previous one and use it to remove the original one from the list. To remove a *Grocery* automatically you need to create the *equals* method in the *Grocery* class. Specify that two *Grocery* are the same if their *name* is the same.
- Add one of the existing groceries twice and display the list.
- Remove the third element of the list by providing its position and display it.
- Display its size.

Hints:

➢ *Use Intellij shortcuts to create the toString and equals methods.*

```
Groceries: []
Groceries: [name=bananas, name=oranges, name=tomatoes]
Groceries: [name=bananas, name=tomatoes]
Groceries: [name=bananas, name=tomatoes, name=tomatoes, name=tomatoes]
Groceries: [name=bananas, name=tomatoes, name=tomatoes]
```

```
Groceries size: 3
```

Exercise 6 [data structures - sets]

Hansel is preparing himself for a trip and wants to make sure he has packed at least one of everything.
Create the *AdvancedBackpackApplication* class and write the following instructions in its *main* method:

- Create the *Item* class that has one *name*.
- Create an empty *backpack* set of *Item* and display it.
- Create three different *Item* objects.
- Add them to the *backpack* set and display it. To display the *Item* class create the *toString* method.
- Create a new *Item* with the same *name* as a previous one and use it to ask if is contained in the set and display the answer. To compare an *Item* automatically you need to create the *equals* method in the *Item* class. Specify that two *Items* are the same if their *name* is the same.
- Use the last *Item* you created to remove the original one from the set. To remove an *Item* automatically you need to create the *equals* method in the *Item* class.
- Ask if the element you just removed is not contained in the set and display the answer.
- Add one of the existing items twice and display the set.
- Display its size.

Hints:

➢ *Use Intellij shortcuts to create the toString and equals methods.*

```
Backpack: []
Backpack: [name=toothpaste, name=towel, name=underwear]
The backpack contains underwear
Backpack: [name=toothpaste, name=towel]
The backpack no longer contains underwear
Backpack: [name=toothpaste, name=towel]
Backpack: [name=toothpaste, name=towel]
Backpack size: 2
```

Exercise 7 [data structures - maps]

Matilda just bought a new book about social impact and as she opens it, she sees the index
Create the *AdvancedIndexApplication* class and write the following instructions in its *main* method:

- Create the *Topic* class that has a *name.*
- Create an empty *index* map connecting Integer for the page numbers and a *List of Topic* for the several book topics that will appear in each page and display it.
- Create two Integers that represent two different page numbers.
- Create three *Topic* objects.
- Create two lists: *topics1* and *topics2*.
- Add one *Topic* to *topics1* and two into *topics2*.
- Put the page numbers and *topics1 and topics2* respectively into the *index* map .
- Display the map keys. To display the *Topic* class create the *toString* method.
- Display the map values. To display the *Topic* class create the *toString* method.
- Create another different *Topic*.
- Get the *topics1* from the map and call it *stored*. Add to *stored* the last *Topic* you created. Display the map.
- Ask if one of the keys is contained in the map and display the answer.
- Get the *topics2* from the map and call it *stored2* and display it.
- Display its size.

Hints:

➢ *Use Intellij shortcuts to create the toString and equals methods.*

```
Index: {}
Index: {1=[text=connecting], 2=[text=possitive things, text=ecologic economy]}
Index keys: [1, 2]
Index values: [[text=connecting], [text=possitive things, text=ecologic economy]]
Index entry sets: [1=[text=connecting], 2=[text=possitive things, text=ecologic economy]]
Index: {1=[text=connecting, text=teaching methods], 2=[text=possitive things,
text=ecologic economy]}
The page number 1 exists as key
The topics in page number 2 are [text=possitive things, text=ecologic economy]
Index size: 2
```

Exercise 8 [optionals]

Christmas is coming and Derek’s new working colleges want to organize an “invisible friend” present giving in their company.
Create the *AdvancedGiftApplication* class and write the following instructions in its *main* method:

- Create the *Box* class that has a String called *content*.
- Create an empty *giftBox* Optional of *Box* and display it.
- Create a *Box* with a present name as *content*.
- Assign an Optional of *Box* to the *giftBox* containing the previous *Box* and display it.
- Ask if the *giftBox* is present and display the answer.
- Get the value contained in *giftBox* and display it. To display the *Box* class create the *toString* method.

Hints:

- ➢ *Use Intellij shortcuts to create the toString and equals methods.*

```
Gift: Optional.empty
Gift: Optional[content=Robotoy]
There is a present in the gift box
The gift is a content=Robotoy
```

Exercise 9 [files, test]

Matilda is at a copy shop where the customers use the printers themselves. She wants to make sure she knows how to print one test page.
Follow the instructions below:

- You need the following classes: *FileReader* and *LoremIpsumApplication.*
- You need the following file: *lorem-ipsum.txt*
- Create the *FileReader* class that has the *asLines* method that receives the *filePath* of a file, reads it and returns a list of String representing all the lines of that file.
- In the *LoremIpsumApplication* class *main* method, create a *FileReader*, read all the lines of the *lorem-ipsum.txt* file and display them.
- Create tests for the *FileReader* class.

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Exercise 10 [files, test]

Matilda decides to print now the topics she has in the file.
Follow the instructions below:

- You need the following classes: *FileReader, TopicReader* and *TopicReaderApplication.*
- You need the following file: *topics.txt*
- Reuse the *FileReader* class from exercise 9. Don’t copy it, just reuse it by importing it.
- Create the *TopicReader* class that has a *FileReader* as an attribute. It has the *getTopics* method that uses the *File Reader* to read the *topics.txt* file, transform every line into one *Topic* and returns the list of topics.
- In the *TopicReaderApplication* class *main* method, create a *TopicReader*, use the *getTopics* method and place them inside a list. Display the list of topics. Create a set of *Topic*, add the elements of the previous list and display it to notice that there happens to be duplicates. To ensure uniqueness you have to create the

equals method in the *Topic* class. Do this and run the program again to make sure the set does not display any duplicates.
- Create tests for the *TopicReader* class.

```
Topic list: text=Christmas markets under the snow
Topic list: text=Winter is coming
Topic list: text=Practice makes mastery
Topic list: text=Working over-hours, is it worth it?
Topic list: text=Winter is coming
Topic list: text=Your next holiday destination
Topic list: text=Winter is coming
Topic set: text=Working over-hours, is it worth it?
Topic set: text=Practice makes mastery
Topic set: text=Winter is coming
Topic set: text=Christmas markets under the snow
Topic set: text=Your next holiday destination
```

Exercise 11 [data structures, files, business intelligence]

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 file *world-happiness-2017.csv* in the GitHub repository.
- Reuse the *FileReader* class from exercise 9. Don't copy it, just reuse it by importing it.
- Create the *HappinessRecord* class that has a String *country*, an Integer *rank* and a Double *score*.
- Create the *HappinessApplication class* that has a *main* method that uses the *FileReader* to read the *world-happiness-2017.csv*, transforms its list of lines into a list of *HappinessRecord*, sorts it by *rank* and displays the first five.

```
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 12 [data structures, files, business intelligence]

After hearing for years that *Winter is coming*, it was about time that someone would say *Winter has come*. Many years have passed, many books of *Game of Thrones* have arrived and many of its characters have died. The time has come to ask the right questions, code our application and become the expects of the *Game of Thrones* with its answers.
Develop the code that answers the following questions:

- How many characters appear in the books in total?
- How many characters died?
- Display the overall percentage of men and women that died in all books.
- Which book has the biggest death count with how many?
- Who died in that book?

Follow the points below:

- Use the file *got-characters.csv* in the GitHub repository.
- Reuse the *FileReader* class from exercise 9. Don't copy it, just reuse it by importing it.
- Create the *Character* class that has a String *name*, a String *bookOfDeath* and a String *gender*.
- Create the *CharacterReader* class that has the *getCharacters* method that receives a *filePath,* uses a *FileReader* to read its lines, transforms them into *Characters* and returns them.
- Create the *GotApplication class* that has a *main* method that uses the *CharacterReader* to read the *got-characters.csv* and performs the code that answers the previously given questions.

```
Number of characters: 917
Number of dead characters: 307
```

```
Dead men: 271 | Dead women: 36
Dead men: 88% | Dead women: 11%
Book: 3 | Deaths: 97
Book: 2 | Deaths: 73
Book: 5 | Deaths: 61
Book: 1 | Deaths: 49
Book: 4 | Deaths: 27
Book with the most dead counts is book 3 with 97 deaths
Characters who died in book number 3:
{name : Aegon Frey (Jinglebell) | book of death : 3 | gender : man}
{name : Alyn | book of death : 3 | gender : man}
{name : Anvil Ryn | book of death : 3 | gender : man}
{name : Bannen | book of death : 3 | gender : man}
{name : Becca the Baker | book of death : 3 | gender : woman}
…
```