Java - elements of functional programming (`II`)

Working environment setup

1. Download and unzip `lab06` source code
    1. Download `lab07.zip` from the course site (moodle)
    2. Unzip it (you get `lab07` directory)
    3. Move `lab07` to `programming-in-java` directory, i.e.,
        - `programming-in-java`
            - `lab00`
            - ...
            - `lab07` <--
            - `gradle`
            - ...
2. [`IntelliJ`] Add `lab07` module to the `programming-in-java` project
    1. In the *Project* window click `settings.gradle` file to open it
    2. Modify its content to the following form:

        ```
        rootProject.name = 'programming-in-java'
        include 'lab00'
        ...
        include 'lab06'
        include 'lab07'
        ```

    3. Save the file
    4. Click `Load Gradle Changes` (a small box in the top right corner)

1) Streams (finite and infinite) creation

Analyse the source code in package `lst07_01`

Exercises

1. Explain the concept of a *stream* (as defined in java.util)
2. Familiarize yourself with the static methods of the Stream interface
3. Define a finite stream of 5 boolean values
4. Define an infinite stream of random integer values
5. Define the infinite stream of even positive integer values
6. [*optional*] Define the stream of the first 20 prime numbers
7. [*optional*] Define the infinite stream of Fibonacci numbers

2) Stream methods (`I`): `skip` , `peek` , `takeWhile` , `dropWhile` , `distinct` , `sorted` , `max` , `min` , `count` , `findFirst` , `findAny` , `anyMatch` , `allMatch` , and `noneMatch`

Analyse the source code in package `lst07_02`

Exercises

1. Familiarize yourself with the instance (non-static) methods of the Stream interface
2. Partition the following methods into intermediate and terminal:
   distinct, *limit*, *skip*, *count*, *max*, *min*, *findFirst*, *findAny*, *allMatch*, *anyMatch*, *noneMatch*, *forEach*, *peek*, *takeWhile*, *dropWhile*, *sorted*
3. Explain potential applications of method *peek*
4. Explain the difference between methods *skip* and *dropWhile*
5. Explain the rule of chaining methods *distinct* and *sorted* (compare the two possible orderings, i.e, "*distinct -> sorted*" vs. "*sorted* -> "*distinct*)

3) `throw-catch` and functional programming mismatch

Analyse the source code in package `lst07_03`

Exercises

1. Explain the problem of the " `throw-catch` and functional programming mismatch"
2. Compare the way of handling *checked* and *unchecked* exceptions in the Java stream pipelines (and lambda expressions)
3. Explain the advantages of using *Optional* when compared to throwing excpetions or using the *null*)

4) Stream methods (`II`): `filter` , `map` , `flatMap` , and `reduce`

Analyse the source code in package `lst07_04`

1. Write imperative (loop based) implementations of `filter` , `map` , `flatMap` , and `reduce`
2. Given a stream of 100 random integers print out only even values
3. Given:

    ```
    List.of("alpha", "bravo", "charlie", "delta")
    ```

    print out the first letter (capitalized) of each element
4. Using `flatMap` , flatten the following list:

    ```
    List.of(List.of(1, 2), List.of(3, 4, 5), List.of(6, 7, 8, 9)
    ```

5. Given a stream of 100 random (positive) integers, using the method `reduce` compute their sum
6. Given a stream of 7 random (positive) integers, using the method `reduce` compute their product
7. Given a stream of 100 random (positive) integers, using the method `reduce` find their max value
8. Given a stream of 10 random (positive) integers, using the method `reduce` concatenate them

Exercises

5) Primitive type streams: `IntStream` , `LongStream` , and `DoubleStream`

Analyse the source code in package `lst07_05`

Exercises

1. Compare `IntStream` with `Stream<Integer>`
2. Explain the output of the following code:

```
int[][] a = {{1, 2}, {3, 4}, {5, 6}};
System.out.println(Stream.of(a)
    .mapToInt(e -> IntStream.of(e).sum())
    .sum());

double[] numbers = {1.2, 1, 2.2, 3.6};
System.out.println(DoubleStream.of(numbers)
  .mapToInt(e -> (int)e).sum());
```

3. Explain the output of the following code:

```
System.out.println(Stream.of(new Character[] {'D', 'B', 'A', 'C'})
            .mapToInt(e -> e - 'A').sum());
```

4. Create a stream of 1000 random integers and then calculate their *min*, *max*, *sum* and *average*
5. Given a finite stream of strings, find the average string length

6) Stream pipelines and collectors

Analyse the source code in package `lst07_06`

Exercises

1. Given:

```
record Client(String name, String address, Optional<String> email) {}
record Ticket(String departure, String destination, LocalDate date, Clien
t client, int priceInUnits) {}
```

   generate a list of clients (i.e., `List<Client> clients`) and and a list of tickets (i.e., `List<Ticket> tickets`)

2. Given a list of tickets and a destination, compute the number of tickets with the given destination
3. Given a list of tickets and a date, print out the tickets for the given date
4. Given a list of tickets and the name of a client, check if there is at least one ticket reserved fot the given client
5. Given a list of tickets return the average value of the prices for all the tickets in the list
6. Given a list of tickets check if all the clients have an email address
7. Given a list of tickets return return a comma separated value containing all the destination in the list

7) Streams in text processing (`I`)

Analyse the source code in package `lst07_07`

Exercises (related to the IMDB top 250 movies)

1. Compute the total number of actors
2. Compute the total number of movies rated "PG-13"
3. Compute the total number of genres
4. Compute the list of movies for each certification (i.e, `"R" => ["The Shawshank Redemption", "The Godfather",...], "PG-13" => ["The Dark Knight", "Forrest Gump, ... ], ...`)
5. Compute the number of movies for each certification
6. Compute the list of movies for each actor (i.e., `"Morgan Freeman" => ["The Shawshank Redemption", "Se7en", ...]`)
7. Compute the number of movies for each actor
8. Compute 5 most frequent directors (sorted)
9. Compute 5 most actors (sorted)

8) Streams in text processing (`II`)

Analyse the source code in package `lst07_08`

Exercises (related to the text of "Alice's Adventures in Wonderland")

1. Compute the total number of words
2. Compute the total number of italicized words (i.e., `_it_`)
3. Compute the number of words for each chapter
4. Compute 10 most frequent words in the whole text (sorted)
5. Compute 10 most frequent words for each chapter (sorted)
6. Compute 10 longest words in the whole text (sorted)
7. Compute 10 longest words for each chapter (sorted)
8. Compute the frequency table of vowels in the whole text
9. Compute the frequency table of vowels for each chapter

9) Push the commits to the remote repository