Java Workbook

04. Stream API, Vol. I

~by Andrzej "Andret2344" Chmiel

Exercise 1: List printout

Given the following list:

```
List.of("Java", "C++", "Python", "C#", "JavaScript");
```

Using the Stream API, print out all the strings from the list. Don't use any loops.

Exercise 2: Population

Given the following code:

```
enum Country {
    POLAND, GERMAN
}

enum Hobby {
    DANCE, TENNIS;
}

record Person(String name, double salary, Country country, Hobby hobby) {
}

class MainPopulation {
    public static void main(final String[] args) {

    }

    private static List<Person> getFixedpopulation() {
        return List.of(
                new Person("Jan", 1900, Country.POLAND, Hobby.DANCE),
                new Person("Marek", 2600, Country.POLAND, Hobby.DANCE),
                new Person("Paul", 4000, Country.GERMAN, Hobby.TENNIS),
                new Person("Jan", 500, Country.GERMAN, Hobby.DANCE),
                new Person("Tom", 3000, Country.GERMAN, Hobby.TENNIS));
    }
}
```

In the `main` method, using the `getFixedPopulation()` method:

1. Print all the people.
2. Print all the people that like to dance.
3. Count how many Germans like to dance.
4. Print the German who earns the most.
5. Sort the people by their earnings.
6. Sort the people alphabetically.
7. Group the people by their nationality.

Exercise 3: Functional interfaces

Given the following code:

```
class Employee(String name, int height, double salary) {
}
```

Not using any loops:

1. Create a `Supplier` that creates a random `Person` with the name's length between 6 and 9 (first char should be uppercased) and with the height between 150 and 220 centimeters and the earnings between 500 and 5000.
2. Create a list of 10 random `Person` objects.
3. Print out those 10 objects sorted alphabetically by name in reverse order.
4. Using the list created earlier, create a map that maps the name to the salary.
5. Update people from the map from the point above, giving 10% pay rise to everyone.

Exercise 4: Reading a book

From the website www.wolnelektury.pl, download any book's content and save it as a `book.txt` file inside the project.

1. Create a method that creates a `Stream` with words from the file (ignore all the punctuation marks).
2. Using the method created above (convert all the letters to lowercase), create a `Map` that maps each word to its occurrences' count. Print the map sorted by the count.
3. Group all the words into two separate groups: those that contain the `'a'` letter, and those that don't.
4. Group all the words by their length.
5. Check if there is a word longer than 15 chars that starts with an uppercase letter. Print any of those words if found any, or `"No such a word"` otherwise.
6. Check if all the words contain only uppercase and lowercase letters, and print the appropriate information.

Exercise 5: List filtering

Given the following code:

```
record Product(String name, String type, double price) {
}

class MainProducts {
    public static void main(final String[] args) {
        final List<Product> productList = List.of(
            new Product("eggs", "food", 7.99),
            new Product("bike", "other", 1299.89),
            new Product("beer", "drink", 3.5),
            new Product("ham", "food", 4.85),
            new Product("apples", "food", 2.49),
            new Product("butter", "food", 6.99),
            new Product("cake", "food", 23.39));
        final double avg = countAverage(productList);
        System.out.printf("Average price of food: %.2f", avg);
    }

    private static double countAverage(final List<Product> productList) {
        return 0;
    }
}
```

Using the Stream API, fill in the `countAverage` method so that it counts an average price of the given products. So running the code prints this:

```
Average price of food: 9.14
```

Exercise 6: The most lowercase letters

Given the following code:

```
List.of("Smith", "Richard", "Uh-oh", "Defenestration");
```

In the list, find the string that has the most lowercase letters. For debugging purpose for each word, print out its lowercase letters count.

Exercise 7: Exceptional lambda expressions

Given the following code:

```
class Main {
    private static final ToIntFunction<Path> function = Main::countLines;

    public static void main(final String[] args) {
        Stream.of("anyFile1", "anyFile2", "anyFile3")
                .map(Paths::get)
                .mapToInt(function::applyAsInt)
                .sorted()
                .forEach(System.out::println);
    }

    private static int countLines(final Path path) throws IOException {
        return Files.readAllLines(path).size();
    }
}
```

Fix the code, so it compiles and attempts to read lines count from all the given files and does not print any exceptions. Do not change the `countLines` method and the value of the `function` field. You can modify the stream, but keep its general purpose. Do not create any new methods.

Exercise 8: Work with streams

Given the following code:

```
final class Main {
    public static void main(final String[] args) {
        final IntStream stream1 = IntStream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 21);
        final IntStream stream2 = IntStream.of(1, 3, 5, 7, 9, 11, 13, 15, 17, 19);
        workWithStreams(stream1, stream2).forEach(System.out::println);
    }

    private static Stream<String> workWithStreams(final IntStream intStream1,
                                                  final IntStream intStream2) {
        return Stream.empty();
    }
}
```

Fill in the `workWithStreams` method so it contains exactly one statement and running the code prints out this:

```
1=1
3=9
5=25
7=49
9=81
11=121
13=169
15=225
17=289
19=361
21=441
```

Exercise 9: Limited filter

Given the following code:

```
final class LimitedFilter {
    private static List<Integer> limitedFilter(final IntStream intStream,
                                               final IntPredicate predicate,
                                               final int m,
                                               final int n) {
        return Collections.emptyList();
    }

    public static void main(final String[] args) {
        final Supplier<IntStream> myStream = () -> IntStream.iterate(1, i -> i + 1);
        final IntPredicate myPredicate = i -> i % 2 == 0;
        System.out.println(limitedFilter(myStream.get(), myPredicate, 3, 10));
        System.out.println(limitedFilter(myStream.get(), myPredicate, 7, 11));
        System.out.println(limitedFilter(myStream.get(), myPredicate, 13, 12));
    }
}
```

Fill in the `limitedFilter` method so it returns a list of integers from the provided stream that matches the given predicate. Only the first `m` elements should be checked, and the result list can't contain more than `n` elements, so running the code prints out this:

```
[2, 4, 6]
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10, 12]
```

Exercise 10: What can a stream do?

Create streams that does following tasks using the provided code.

I.
1. List all company names.
2. For each company on the list, list the number of employees.
3. Only list companies that have an odd employee-count.
4. Sum up all employees from all the companies.
5. Calculate the average number of employees for all the companies listed.
6. List the company names and the employee count alphabetically.

II.
1. List all the male employees of all companies.
2. Only list the first and last names of the non-male employees.
3. Only list companies that have more males than females.
4. List the average salary of each company individually.
5. List all employees earning more than 3400.
6. List all employees in order of their salary, descending.
7. (*) For each company, display the gender, for which the average salary is greater.

III.
1. List all employees who have the HR or the PR role.
2. List companies that have more than 2 employees with the WORKER role.
3. Sum up the salary of all employees from all the companies that have the MANAGER role.
4. Count the average salary of all employees having more than 1 role.
5. List all the employees with a PR role but without the MANAGER role.
6. List last names and salaries only of the employees having the ACCOUNTANT role.
7. For each company, count all the roles assigned to its employees.

```java
enum Role {
    WORKER, MANAGER, HR, ACCOUNTANT, PR
}

enum Gender {
    MALE, FEMALE, NON_BINARY
}

record Employee(String firstName, String lastName, Gender gender,
                List<Role> roles, double salary) { }

record Company(String name, List<Employee> employees) { }

final class MainStreaming {
    public static void main(final String[] args) {
        final List<Company> companies = List.of(
                new Company("SDA", List.of(
                        new Employee("Tom", "Johnson", Gender.MALE,
                                List.of(Role.HR, Role.ACCOUNTANT), 3000),
                        new Employee("Maria", "Rose", Gender.FEMALE,
                                List.of(Role.HR, Role.ACCOUNTANT), 3200),
                        new Employee("Rob", "Williams", Gender.NON_BINARY,
                                List.of(Role.WORKER, Role.PR), 3500),
                        new Employee("John", "Smith", Gender.MALE,
                                List.of(Role.MANAGER), 3900),
                        new Employee("Caren", "Ground", Gender.FEMALE,
                                List.of(Role.WORKER, Role.PR), 4100),
                        new Employee("Jerry", "Williams", Gender.MALE,
                                List.of(Role.PR, Role.WORKER), 4100)
                )),
                new Company("Avengerex", List.of(
                        new Employee("Steven", "Strange", Gender.MALE,
                                List.of(Role.PR), 3600),
                        new Employee("Natasha", "Romanow", Gender.FEMALE,
                                List.of(Role.WORKER, Role.PR), 4200),
                        new Employee("Loki", "Asgardian", Gender.NON_BINARY,
                                List.of(Role.PR, Role.MANAGER), 3800),
                        new Employee("Tony", "Stark", Gender.MALE,
                                List.of(Role.ACCOUNTANT, Role.PR), 4500),
                        new Employee("Carol", "Danvers", Gender.FEMALE,
                                List.of(Role.WORKER, Role.HR), 4100))));
        System.out.println(companies);
    }
}
```