Java Workbook

03. Functional programming

~by Andrzej "Andret2344" Chmiel

Exercise 1: List reducing

Given the following code:

```
class MainReducer {
    public static void main(final String[] args) {
        final BinaryOperator<Integer> sum = null;
        final BinaryOperator<Integer> sumOfSquares = null;
        final BinaryOperator<Integer> sumOfCubes = null;
        final List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6);
        System.out.println(reduce(list, sum, 0));
        System.out.println(reduce(list, sumOfSquares, 0));
        System.out.println(reduce(list, sumOfCubes, 0));
    }

    private static <E> E reduce(final List<E> list,
                                final BinaryOperator<E> operator,
                                final E initialValue) {
        return initialValue;
    }
}
```

Fill in the `reduce` method and variables initialization, so the program calculates correctly the sum, the sum of squares and the sum of cubes of the given list of numbers and the output looks like this:

```
21
91
441
```

Exercise 2: Single function and operand

Given the following code:

```
class MainSingleFunctionAndOperand {
    public static void main(final String[] args) {
        oneFunction.accept(
                Arrays.asList(
                        new Pair<>(1, 2),
                        new Pair<>(4, 5),
                        new Pair<>(1, 10)),
                Integer::sum);

        oneOperand.accept(
                new Pair<>(10, 2),
                Arrays.asList(
                        (a, b) -> a - b,
                        (a, b) -> a * b,
                        (a, b) -> a / b));

    }
}
```

Create fields `oneFunction` and `oneOperand` in the given class so that the code compiles and prints out this:

```
3
9
11
8
20
5
```

Exercise 3: Functional interfaces

Given the following code:

```
class MainFunctionalInterfaces {
    public static void main(final String[] args) {
        // Create variables here
        System.out.println("2 + 2 = " + sum.apply(2, 2));
        System.out.println("3 is " + (ifPrime.test(3) ? "prime" : "nonprime"));
        System.out.println("4 is " + (ifPrime.test(4) ? "prime" : "nonprime"));
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
        System.out.println("Next prime number: " + nextPrime.get());
    }
}
```

In the pinpointed place initialize variables `sum`, `ifPrime` and `nextPrime` so the code compiles and prints out this:

```
2 + 2 = 4
3 is prime
4 is nonprime
Next prime number: 2
Next prime number: 3
Next prime number: 5
Next prime number: 7
Next prime number: 11
Next prime number: 13
Next prime number: 17
Next prime number: 19
```

Exercise 4: List printout

Given the following list.

```
List<String> list = List.of("dog", "cat", "parrot", "mouse", "turtle");
```

Not using any loop, print out elements of the list uppercase and in separate lines.

Exercise 5: List filtering

Given the following code:

```
class MainListFilter {
    public static void main(final String[] args) {
        final List<String> strings = List.of("dog", "cat", "mouse", "magic");
        System.out.println("Texts longer than 3 chars:");
        filter(strings, /* fill it in */ );
        System.out.println("Texts containing 'a':");
        filter(strings, /* fill it in */ );
    }
}
```

Fix the code so it compiles and prints out this:

```
Texts longer than 3 chars:
mouse
magic
Texts containing 'a':
cat
magic
```

Exercise 6: Closure

Given the following code:

```
class MainClosure {
    public static void main(final String[] args) {
        /* HERE */
        System.out.println(func.apply(5).apply(2));
        System.out.println(func.apply(5).apply(3));
        final Function<Integer, Integer> f = func.apply(100);
        System.out.println(f.apply(13));
        System.out.println(f.apply(-100));
    }
}
```

In the pinpointed place add a declaration of the variable `func` so the code compiles and prints out this:

```
7
8
113
0
```

To compile the code, do not change any of the existing lines of code. You can improve the code after it compiles and works properly.

Exercise 7: Map printout

Given the following code:

```
class MainMapPrinting {
    public static void main(final String[] args) {
        final Map<String, Integer> map = Map.ofEntries(
                Map.entry("Java", 18),
                Map.entry("Python", 1),
                Map.entry("PHP", 0));
        printMap(map);
    }

    private static void printMap(final Map<String, Integer> map) {
    }
}
```

Fill in the `printMap` method, so the output looks like this:

```
Key: Java, value: 18,
Key: Python, value: 1,
Key: PHP, value: 0.
```

Pay attention to the ending character, the last line uses a full stop instead of a comma.

Exercise 8: Percentage

Create a generic method `partOf` that accepts a list or an array and a matching predicate and counts how many elements match to the given condition and returns a value between `0` and `100`. So running the following code:

```
class MainPercentage {
    public static void main(final String[] args) {
        final List<Integer> list1 = List.of(1, 2, 3, 4, 5, 6, 7, 8, 9);
        System.out.println(partOf(list1, i -> i % 2 == 0));
        System.out.println(partOf(list1, i -> i > 7));
        System.out.println(partOf(list1, i -> false));

        final List<Integer> list2 = List.of(1, 10, 100, 1000, 10000);
        System.out.println(partOf(list2, i -> i % 10 == 0));
        System.out.println(partOf(list2, i -> (i + 1) % 11 == 0));
    }
}
```

prints out this:

```
50.0
20.0
0.0
80.0
40.0
```

Exercise 9: CSV parser

Given the following code:

```
record Person(String firstName, String lastName, int age) {
}

class MainCSVParser {
    public static void main(final String[] args) {
        final String s = "Name;Surname;Age\n" +
                "Ignacy;Krasicki;57\n" +
                "Adam;Mickiewicz;44\n";
        final List<Person> personList = parseCSV(s);
        for (final Person person : personList) {
            System.out.println(person);
        }
    }

    private static List<Person> parseCSV(String input) {
        return Collections.emptyList();
    }
}
```

Fill in the `parseCVS` method so the output looks like this:

```
Person[firstName=Ignacy, lastName=Krasicki, age=57]
Person[firstName=Adam, lastName=Mickiewicz, age=44]
```

Exercise 10: Length converter

Given the following code:

```
enum UnitConverter {
    METERS_TO_YARDS,
    YARDS_TO_METERS,
    CENTIMETERS_TO_INCHES,
    INCHES_TO_CENTIMETERS,
    KILOMETERS_TO_MILES,
    MILES_TO_KILOMETERS;

    public double convert(final double input) {
        return input;
    }
}

class MainCSVParser {
    public static void main(final String[] args) {
        System.out.println(UnitConverter.METERS_TO_YARDS.convert(10));
        System.out.println(UnitConverter.YARDS_TO_METERS.convert(10));
        System.out.println(UnitConverter.CENTIMETERS_TO_INCHES.convert(10));
        System.out.println(UnitConverter.INCHES_TO_CENTIMETERS.convert(10));
        System.out.println(UnitConverter.KILOMETERS_TO_MILES.convert(10));
        System.out.println(UnitConverter.MILES_TO_KILOMETERS.convert(10));
    }
}
```

Changing only the enum, fix the given code, so it prints out this:

```
10.93999981880188
9.100000262260437
3.8999998569488525
25.0
6.200000047683716
16.100000143051147
```

Do not use any generic type.