Java - elements of functional programming (I)

Working environment setup

1. Download and unzip `lab06` source code
    1. Download `lab06.zip` from the course site (moodle)
    2. Unzip it (you get `lab06` directory)
    3. Move `lab06` to `programming-in-java` directory, i.e.,
        - `programming-in-java`
            - `lab00`
            - ...
            - `lab06` <--
            - `gradle`
            - ...
2. [`IntelliJ`] Add `lab06` 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 'lab05'
include 'lab06'
```

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

1) Functional interfaces, lambda expressions and method references

Analyse the source code in package `lst06_01`

Exercises

1. Explain the following concepts: *functional interface*, *lambda expression*, and *method reference*
2. Write and test anonymous functions (lambda expressions) corresponding to:
    - $f_1(x) = x - 2;\ x \in \mathbb{R}$
    - $f_2(x, y) = \sqrt{(x^2 + y^2)};\ x, y \in \mathbb{R}$
    - $f_3(x, y, z) = \sqrt{(x^2 + y^2 + z^2)};\ x, y, z \in \mathbb{Z}$
3. Write and test anonymous functions corresponding to:

```
sqrt, abs, log, id
```

4. Given:

```
@FunctionalInterface
interface FunIf<T, R> {
    R apply(T t);
}
```

   complete the following code using lambda expressions:

```
FunIf<String, Integer> f1 = ___;
FunIf<Integer, String> f2 = ___;
FunIf<Double, Double> f3 = ___;
FunIf<Integer, Boolean> f4 = ___;
FunIf<Boolean, Integer > f5 = ___;
FunIf<Boolean, Boolean > f6 = ___;
```

5. Repeat the previous exercise using method references instead of lambda expressions (*note*: you should probably implement these methods first)

2) Standard functional interfaces

Analyse the source code in package `lst06_02`

Exercises

1. Familiarize yourself with the functional interfaces available in `java.util.function` package
2. For each of the standard functional interfaces white at least one example that demonstrates its use, i.e.:

```
BiConsumer<String, String> bc = (s1, s2) -> System.out.println(s1 + " " +
s2);
BiFunction... = (..., ...) -> ...
BinaryOperator...
...
```

3. Explain the rationale behind the primitive type specialisations of the standard generic functional interfaces (e.g., `BooleanSupplier` , `DoubleConsumer`)

3) Higher-order functions

Analyse the source code in package `lst06_03`

Exercises

1. Using `sumOfWith` , without defining any new functions, calculate $\sum_{i=1}^{15} i^5$
2. Write and test function

```
DoubleUnaryOperator expApproxUpTo(int n) {
```

```
    //...
}
```

that returns the n-th order ($n < 6$) Maclaurin polynomial of the function e^x, i.e. `expApproxUpTo(n)` $= \sum_{k=0}^{n} \frac{x^k}{k!}$.

3. Write and test function

```
DoubleUnaryOperator dfr(DoubleUnaryOperator f, double h) {
    //...
}
```

that returns for a given function f, the approximation of its first derivative f' calculated as (finite-difference): $f'(x_0, h) \approx \frac{f(x_0+h)-f(x_0)}{h}$.

Check the approximation errors corresponding to different values of `h`

4. (*optional*) Write and test function

```
DoubleUnaryOperator d2f(DoubleUnaryOperator f, double h) {
    //...
}
```

that returns for a given function f, the approximation of its second derivative f''; use different finite-difference schemes

5. Analyse and test the following method:

```
private static <T, R> List<R> applyAll(List<Function<T, R>> fs, T x0) {
    List<R> ys = new ArrayList<>();

    for (var f : fs) {
        ys.add(f.apply(x0));
    }

    return Collections.unmodifiableList(ys);
}
```

4) Function composition

Analyse the source code in package `lst06_04`

Exercises

1. Using `Function.compose` create $(f_i \circ g_i)(x)$ for the following pairs of f and g:
   - $f_1(x) = 2x,\ g_1(x) = x^2,\ x \in \mathbb{R}$
   - $f_2(x) = sin(x),\ g_2(x) = \frac{1-x}{1+x^2},\ x \in \mathbb{R}$
   - $f_3(x) = \frac{1-sin(x)}{1+2x^2},\ g_3(x) = cos(x),\ x \in \mathbb{R}$
2. Repeat the previous exercise using `Function.andThen`
3. (*optional*) Write a function/method that composes a given list of functions

5) Dealing with optional data

Analyse the source code in package `lst06_05`

Exercises

1. Familiarize yourself with class Optional
2. Describe pros and cons of the following approaches to represent a "no-valid-result" of a function/method:
   - throwing an exception
   - returning null
   - using Optional
3. Write three variants of a method that returns the tail of a given list (see `headOf_v1`, `headOf_v2`, `headOf_v3` in `lst01_05`)
4. Review the code of `proj1` and identify the methods that could have `Optional` as the return type

6) Push the commits to the remote repository