Test Functional Programming

Polytech SI4 2023-2024

For each exercise, you can (or must in some cases) define additional methods.

***Reminder**: records can have methods like normal classes.*

Question 1)

Given the record Lst : `record Lst<T>(T car, Lst<T> cdr)`, define the method `double average(Lst<Lst<Integer>> l)` that computes the average of a list of lists of integers *in a terminal recursive manner.*

E.g. $average([[1,2,3],[6]]) = \left(\frac{1+2+3+6}{4}\right) = 3$

To compute this average, you should:

- traverse the list of list a single recursive traversal
- compute the sum of the elements as well as the number of elements during this traversal
- compute the average (sum/nb) at the end of the traversal

The result is non-defined in case of an empty list.

Question 2)

Define the method `retainer(T e)` that returns a function that returns the previous value that was used as argument. The first call returns e.

```
f = retainer(10);
System.out.println(f.apply(5));      // 10
System.out.println(f.apply(3));      // 5
System.out.println(f.apply(7));      // 3
```

Question 3)

Define `Stream<Character> toStream(String s)` that returns the stream of the characters of a String. *Do not use methods such as chars() or toCharArray() to transform directly the String into a stream or an array*. You should use the stream creation method iterate (or it's IntStream version).

```
static <T> Stream<T> iterate(T seed, UnaryOperator<T> f)
```

Question 4)

Here is a code, discover what it does, and explain in a few words what each line does, and the reason why it is here.

```
public record SN(int s, int n) {    }

public static double mysterious(Stream<IntStream> stream) {
  BinaryOperator<SN> red = (p, q) -> new SN(p.s() + q.s(), p.n() + q.n());
  //mapToObj is a IntStream method that returns an object-valued
  // Stream consisting of the results of applying the given function
  // to the elements of this IntStream.
  Stream<Optional<SN>> sub =
    stream.map(is -> is.mapToObj(x -> new SN( ,  )).reduce(red));
  Optional<SN> res =
    sub.filter(Optional::isPresent).map(Optional::get).reduce(red);
  return res.map(sn ->(double) sn.s()/sn.n()).orElse(Double.NaN);
}
```

Question 5)

Write the record `Option<T>(T value)` that represents an optional value. The inner value will be null in case of an empty option. Write the following methods to obtain the same behavior than the Java Optional class:

- `empty()` (a new empty Option)
- `of(v)` (a new Option with value v)
- `orElse(v)` (returns the value of the Option if present, or the value v if not)
- `map(f)` (applies function f on the value if present, and returns an Option containing the result of this application, otherwise returns an empty Option)