CS2030 Lab #8: Logger

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Logging

Topic Coverage

- Generics
- Function
- Lambda expression
- `map` and `flatMap`

Problem Description

In this lab, we are going to write an *immutable* `Logger` class to handle the context of logging changes to values while they are operated upon. We do this so as to separate the code for logging from the main business logic.

A sample logging session is shown below:

```
jshell> Logger.<Integer>of(5)
$13 ==> Logger[5]

jshell> Logger.<Integer>of(5).map(x -> x + 1)
$14 ==> Logger[6]
5 -> 6

jshell> Logger.<Integer>of(5).map(x -> x + 1).map(x -> x * 2)
$15 ==> Logger[12]
5 -> 6
6 -> 12

jshell> Logger.<Integer>of(5).map(x -> x + 1).map(x -> x * 2).map(x -> x)
$16 ==> Logger[12]
5 -> 6
6 -> 12
12 -> 12
```

Notice that the log of value changes are tracked and output in the form

```
old_value -> new_value
```

Task

Your task is to write a `Logger` class that provides the operations `of`, `map`, `flatMap` and `test`.

You will also write several applications using the `Logger` as solutions to classic computation problems. This would allow us to look at the values changes when solving each problem.

This task is divided into several levels. Read through all the levels to see how the different levels are related.

Level 1

Start by writing a static method `of` within the class `Logger` that wraps a value within it. Include the `toString()` method to output the `Logger` containing the value.

To ensure that a valid `Logger` is created, the `of` method should throw an `IllegalArgumentException` when the argument is `null` or another `Logger`.

```
jshell> Logger.<Integer>of(5)
$.. ==> Logger[5]

jshell> Logger<String> hello = Logger.<String>of("Hello")
hello ==> Logger[Hello]

jshell> try { Logger.<Object>of(hello); }
   ...> catch (Exception e) { System.out.println(e); }
java.lang.IllegalArgumentException: already a Logger

jshell> try { Logger.<Integer>of(null); }
   ...> catch (Exception e) { System.out.println(e); }
java.lang.IllegalArgumentException: argument cannot be null
```

Level 2

Include a `map` method that takes in a function `Function<? super T, ? extends U>`, applies it on the value, and wraps the result in another `Logger`. Modify the `toString()` method to include the output of the value changes over all `map` operations.

```
jshell> Logger<Integer> five = Logger.<Integer>of(5)
five ==> Logger[5]

jshell> five.map(x -> x + 1)
$.. ==> Logger[6]
5 -> 6

jshell> five
five ==> Logger[5]

jshell> five.map(x -> x + 1).map(x -> x - 1)
$.. ==> Logger[5]
5 -> 6
6 -> 5

jshell> five.map(x -> x + 1).map(x -> x - 1).map(x -> x)
$.. ==> Logger[5]
5 -> 6
6 -> 5
5 -> 5

jshell> Logger.<String>of("hello").map(x -> x.length())
$.. ==> Logger[5]
hello -> 5

jshell> Logger.<String>of("hello").map(x -> x.length()).map(x -> x * 2)
$.. ==> Logger[10]
hello -> 5
5 -> 10

jshell> Function<Object, Integer> f = x -> x.hashCode()
f ==> $Lambda$26/0x00000008000bbc40@754ba872

jshell> Logger.<String>of("hello").map(f)
$.. ==> Logger[99162322]
hello -> 99162322

jshell> Function<String, Integer> g = x -> x.length();
g ==> $Lambda$27/0x00000008000bb040@464bee09

jshell> Logger<Number> lognum = Logger.<String>of("hello").map(g)
lognum ==> Logger[5]
hello -> 5
```

Level 3

Write the `flatMap` method that takes a function `Function<? super T, ? extends Logger<? extends U>>`, applies it on the value and wraps the result in another `Logger`. In particular, note the sequence of value changes being output.

```
jshell> Logger<Integer> five = Logger.<Integer>of(5)
five ==> Logger[5]

jshell> five.flatMap(x -> Logger.of(x + 1))
$.. ==> Logger[6]

jshell> five.map(x -> x + 2).map(x -> x * 10)
$.. ==> Logger[70]
5 -> 7
7 -> 70

jshell> five.flatMap(x -> Logger.of(x).map(y -> y + 2)).
   ...>     flatMap(y -> Logger.of(y).map(z -> z * 10))
$.. ==> Logger[70]
5 -> 7
7 -> 70

jshell> Logger.<Integer>of(5).
   ...>     flatMap(x -> Logger.of(x).
   ...>         map(y -> y + 2).
   ...>         flatMap(y -> Logger.of(y).map(z -> z * 10)))
$.. ==> Logger[70]
5 -> 7
7 -> 70

jshell> Function<Object, Logger<Integer>> f = x -> Logger.<Object>of(x).map(y -> y.hashCo
f ==> $Lambda$29/0x00000008000ca040@4cf777e8

jshell> Logger.of("hello").flatMap(f)
$.. ==> Logger[99162322]
hello -> 99162322

jshell> Function<String, Logger<Integer>> g = x -> Logger.<String>of(x).map(y -> y.length
g ==> $Lambda$31/0x00000008000ca840@1ce92674

jshell> Logger<Number> lognum = Logger.<String>of("hello").flatMap(g)
lognum ==> Logger[5]
hello -> 5
```

Level 4

Include an `equals(Object)` method that returns true if the argument `Logger` object is the same as this `Logger`, or false otherwise. Two loggers are equal if and only if both the wrapped value as well as the logs are the same.

```
jshell> Logger<Integer> five = Logger.<Integer>of(5)
five ==> Logger[5]

jshell> Logger.<Integer>of(5).equals(five)
$.. ==> true

jshell> Logger.<Integer>of(5).map(x -> x).equals(five)
$.. ==> false

jshell> five.equals(five)
$.. ==> true

jshell> five.equals(5)
$.. ==> false

jshell> Function<Integer,Logger<Integer>> f = x -> Logger.of(x).map(y -> y + 2);
f ==> $Lambda$18/0x00000008000bec40@57baeedf

jshell> Function<Integer,Logger<Integer>> g = x -> Logger.of(x).map(y -> y * 2);
g ==> $Lambda$19/0x00000008000bd840@5442a311

jshell> Logger.of(5).flatMap(f).equals(f.apply(5)) // left identity
$.. ==> true
```

```
jshell> five.flatMap(x -> Logger.of(x)).equals(five) // right identity
$.. ==> true

jshell> five.flatMap(f).flatMap(g).equals(
   ...>     five.flatMap(x -> f.apply(x).flatMap(g)))
$.. ==> true

jshell> five.flatMap(f).flatMap(g).equals(
   ...>     five.flatMap(x -> f.apply(x).flatMap(g))) // associativity
$.. ==> true
```

Level 5

Let's write some applications using JShell that makes use of our `Logger` so as to observe how the values changes over the course computation. Save your methods in the file `level5.jsh`.

Define an `add(Logger<Integer> a, int b)` method that returns the result of `a` added to `b` wrapped in a `Logger` that preserves the log of all operations of `a`, as well as the addition to `b`.

```
jshell> add(Logger.<Integer>of(5), 6)
$.. ==> Logger[11]
5 -> 11

jshell> add(Logger.<Integer>of(5).map(x -> x * 2), 6)
$.. ==> Logger[16]
5 -> 10
10 -> 16
```

The sum of non-negative integers from `0` to `n` (inclusive of both) can be computed *recursively* using

```
int sum(int n) {
    if (n == 0) {
        return 0;
    } else {
        return sum(n - 1) + n;
    }
}
```

Redefine the above method such that it returns the result wrapped in a `Logger`. You may find the above `add` method useful.

```
jshell> sum(0)
$.. ==> Logger[0]

jshell> sum(5)
$.. ==> Logger[15]
0 -> 1
1 -> 3
3 -> 6
6 -> 10
10 -> 15
```

The *Collatz conjecture* (or the `3n+1` *Conjecture*) is a process of generating a sequence of numbers starting with a positive integer that *seems to always* end with `1`.

```
int f(int n) {
   if (n == 1) {
      return 1;
   } else if (n % 2 == 0) {
      return f(n / 2);
   } else {
      return f(3 * n + 1);
   }
}
```

Write the function `f` that takes in `n` and returns a `Logger<Integer>` that wraps around the final value, with a log of the value changes over time. You should include a `test` method in the `Logger`

```
Logger<T> test(Predicate<? super T> pred, Logger<T> trueLogger, Logger<T> falseLogger)
```

that takes in a `Predicate` and two loggers, and returns the former or latter `Logger` depending on whether `pred` is evaluated to `true` or `false`.

```
jshell> Logger<Integer> five = Logger.<Integer>of(5)
five ==> Logger[5]

jshell> five.test(x -> x == 5, five.map(x -> x + 1), five.map(x -> x - 1))
$.. ==> Logger[6]
5 -> 6

jshell> five.map(x -> x + 1).test(x -> x == 5, five.map(x -> x + 1), five.map(x -> x - 1)
$.. ==> Logger[4]
5 -> 4

jshell> five.map(x -> x + 1).
   ...>     test(x -> x == 5,
   ...>         five.map(x -> { System.out.println("add 1"); return x + 1; }),
   ...>         five.map(x -> { System.out.println("sub 1"); return x - 1; }))
sub 1
$.. ==> Logger[4]
5 -> 4

jshell> f(16)
$.. ==> Logger[1]
16 -> 8
8 -> 4
4 -> 2
2 -> 1

jshell> f(10)
$.. ==> Logger[1]
10 -> 5
5 -> 15
15 -> 16
16 -> 8
8 -> 4
4 -> 2
2 -> 1
```

Terms of Use | Privacy | Non-discrimination

MySoC | Computing Facilities | Search | Campus Map
School of Computing, National University of Singapore