Making Your Program Spicy
Currying
and
Partial Function Application (PFA)

Dhaval Dalal
@softwareartisan
https://dhavaldalal.wordpress.com

14th Nov 2019

Currying

- Refers to the phenomena of rewriting a N-arg function to a nest of functions, each taking only 1-arg at a time

```
// (String, Integer) -> String
String merge(String x, Integer y) {
  return x + y.toString();
}
// String -> (Integer -> String)
Function<Integer, String> merge(String x) {
  return y -> x + y.toString();
}
// () -> (String -> (Integer -> String))
Function<String, Function<Integer, String>> merge() {
  return x -> y -> x + y.toString();
}

Function<String, Function<Integer, String>> merge =
  x -> y -> x + y.toString();

System.out.println(merge("Test#", 1));                  // Test#1
System.out.println(merge("Test#").apply(2));            // Test#2
System.out.println(merge.apply("Test#").apply(3));      // Test#3
```

Currying

- Curried function is a nested structure, just like Russian dolls, takes one arg at a time, instead of all the args at once.

For each arg, there is another nested function, that takes a arg and returns a function taking the subsequent arg, until all the args are exhausted.

$$f : (T, U, V) \mapsto R$$

$$f' : T \mapsto U \mapsto V \mapsto R$$

$$f = f'$$

Currying

- Function type associates towards right.

```
Function<String, Function<Integer, String>> merge =
  x ->
     y ->
        x + y.toString();
```

- In other words, arrow groups towards right.

```
x -> y -> x + y.toString();

// is same as
x -> (y -> x + y.toString());
```

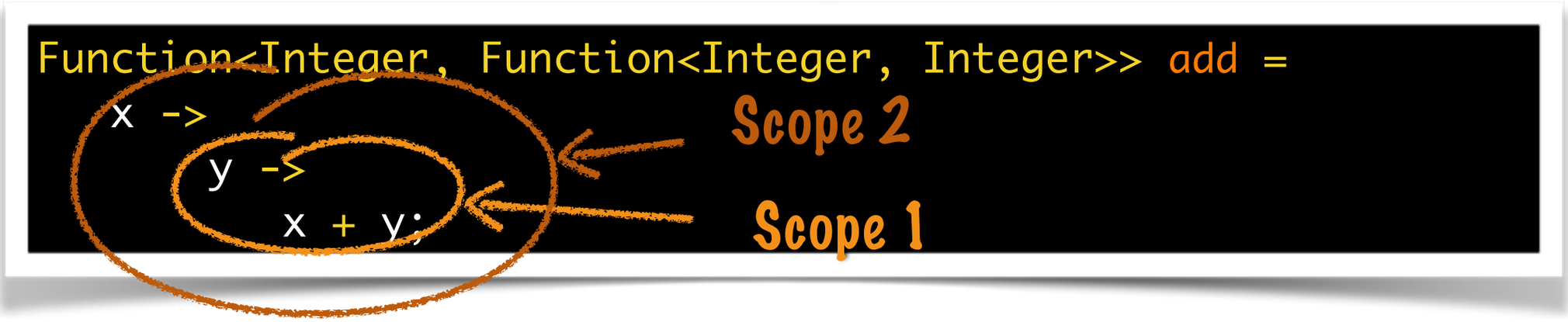
- Function application associates towards left. In other words, apply() groups towards left.

```
merge.apply("Test#").apply(1);

// is same as
(merge.apply("Test#")).apply(1);
```

Why Currying?

- Facilitates arguments to be applied from different scopes.



- Helps us reshape and re-purpose the original function by creating a partially applied function from it

```
Function<Integer, Function<Integer, Integer>> add =
  x ->
     y ->
        x + y;

Function<Integer, Integer> increment = add.apply(1);
increment.apply(2); // 3

Function<Integer, Integer> decrement = add.apply(-1);
decrement.apply(2); // 1
```

Thank - You!

https://github.com/DhavalDalal/FunctionalConference-2019-Currying-PFA-Demo