CSCI 3055 - Assignment 1

Luisa Rojas-Garcia (100518772)

1

Convert the following object oriented code to an implementation that uses closure and functions as values

The source code can be found in the .zip file attached.

Compile and run

```
luisarojas [Assignment1] $ scalac part1.scala
luisarojas [Assignment1] $ scala Main
```

Output

```
28
Clark Kent
29
30
31
```

2

Implement the quicksort algorithm using functional programming. You should only use immutable data structures. Your implementation takes on the form of a function with the following signature:

```
defquicksort[K](comparator:(K,K)=>Int)(input:List[K]):List[K]{
    ...
}
```

The source code can be found in the `.zip` file attached.

Compile and run

```
luisarojas [Assignment1] $ scalac part2.scala
luisarojas [Assignment1] $ scala Main
```

Output

```
BEFORE
List(A, B, X, H, F, Y, Q, A, R)
List(8.0, 34.0, 421.0, 54.0, 1.0, 143.0, 2.0, 65.0, 24.0)
List(8, 34, 421, 54, 1, 143, 2, 65, 24)

AFTER
List(A, B, F, H, Q, R, X, Y)
List(1.0, 2.0, 8.0, 24.0, 34.0, 54.0, 65.0, 143.0, 421.0)
List(1, 2, 8, 24, 34, 54, 65, 143, 421)
```

3

Answer the following questions

3.1 What is the advantage of making `quicksort` a curried function?

Currying, through partially applied functions, enables functions to reduce the number of arguments a certain function takes to one (the other arguments are specified by the actual curry), transforming it into a chain of functions, each taking a single parameter.

This method is very useful, specially for cases such as: function as parameters, by name parameters, type inference and implicit values.

Type inference is improved by using parameters in the first section to infer type parameters that will aid in providing an expected type in the argument section after it. It's important to be able to pull any common functionality out onto higher-order functions.

Lastly, currying also aids turning the code into a more convetient, lighter-weight syntax that is easier to read, understand and follow.

3.2 What would be the Java-equivalent way of achieve the curried version of `quicksort`?

In the case of Java, currying consists of a series of nested functions where the variable types are identified in the following way:

```
public Function<Integer, Function<Integer, Integer>> someFunc() {
    ... nested functions ...
}
```

Following this, parameters will unforld and the function will be overridden. The actual body of the function will go when the nesting is finalized.

Curried functions are called the following way:

```
someFunc().apply(arg1).apply(arg2);
```

It's important to note that Java 8 supports currying, using the new lambda syntax. It would follow the following form:

```
public Function<Integer, Function<Integer, Integer>> someFunc() {
    ... final function body ...
}
```