ScalaDataFrame

We will use the Java code of the first part of the Java assignment to create a layer in Scala that interacts with our DataFrame library.

Review notes:

- The reviews will value both correctness and quality of the code.
- The code should be documented.

You need to use part of the Java classes from the first assignment, so you should be able to import them from your Scala code. For this part, you may only work with CSV files.

1 – COMPOSITE

Reimplement the **composite** pattern in Scala. Leaf objects in Scala can simply be proxy objects to the Java implementation.

Only implement the operations `at`, `columns` and `size` in the Scala composite.

2 – VISITOR

Implement the **visitor** pattern in Scala. Like in the Java part, combine it with the previous composite. The visitors will visit any type of component and the components can accept any visitor. When a node accepts a visitor, it propagates the visit to its children.

Implement a **FilterVisitor** that collects all elements (rows) that fulfill a condition. Like the `query` operation.

```
val v = new FilterVisitor(condition)
rootDF.accept(v)
println("Filtered: " + v.elements)
```

Implement a **CounterVisitor** that counts the number of files and directories it has visited.

```
val c = new CounterVisitor()
rootDF.accept(c)
println("DataFrame files: " + c.files + " DataFrame dirs: " + c.dirs)
```

3 – RECURSION

Add a method to the DataFrame to retrieve one of the columns as a list so that we can apply operations to all values of a column next.

Since the Java DataFrame is likely to return the column as a Java list, we need to do something to transform the list to a Scala list. Import the following to have access to the converters:

```
import scala.jdk.CollectionConverters._
```

And transform the Java list to a Scala one with:

```
javaList.asScala.toList
```

Using **recursion**, implement a function **listFilterMap** that filters the elements in a list based on a given condition and applies an operation to those that fulfill it. The type of the elements is generic, and the function takes three parameters:

- Condition: A=>Boolean
- Operation: A=>B
- Collection: List[A]

- Return: List[B]

Implement listFilterMap using **recursion**. Do it twice: with **stack** recursion and with **tail** recursion.

Use the function to compute the following:

- Round the values of a float-type column that are greater than a value (e.g., 100).
- Replace a certain word from a string-type column on the elements that contain that word.

4– ADVANCED SCALA

Demonstrate **using ScalaDataFrame** the use of the following advanced features of Scala:

- Runtime Multiple Inheritance
  - (https://github.com/pedrotgn/TAP/blob/master/TAPScalaF2/src/MultipleInheritance.scala)
- For loops
  - (https://github.com/pedrotgn/TAP/blob/master/TAPScalaF2/src/forexpression.scala)
- Fold, tabulate
  - (https://github.com/pedrotgn/TAP/blob/master/TAPScalaF/src/ListFunctions.scala)
- Curry, partial parametrization (can be applied on the listFilterMap function)
  - (https://github.com/pedrotgn/TAP/blob/master/TAPScalaF/src/Curry.scala)