О пользе функциональных вставок

Задача: есть слоистая структура, в каждом слое по определенному пути может находится 0, 1 или много элементов. Надо все эти элементы сгруппировать по определенному правилу и в каждой группе слить в один и вернуть список того, что получилось.

В чисто процедурном подходе это можно сделать, но это будет кошмар на сотню строк, при этом, разумеется, никакой истинной универсальности добиться не получится, поскольку нет возможности передать правила по объединению элементов. Максимум, можно запрограммировать несколько фиксированных преобразований.

В чисто объектном подходе можно сделать универсальную реализацию. На каждый вызов надо будет конструировать новый класс, наследующий интерфейс преобразования. В принципе, ничего фатального, но много лишнего класса.

Вот как это выглядит с применением функциональных элементов:

Пример

```
/**
 * Merge node lists grouping nodes by provided classifier and then merging each group independently
 *
 * @param nodeName   the name of node
 * @param classifier grouping function
 * @param collector  used to each group
 * @param <A>        intermediate collector accumulator type
 * @param <K>        classifier key type
 * @return
 */
public <A, K> Collection<Meta> collectNodes(String nodeName, Function<? super Meta, ? extends K> classifier,
Collector<Meta, A, Meta> collector) {
    return layers().stream()
            .filter(layer -> layer.hasMeta(nodeName))
            .flatMap(layer -> layer.getMetaList(nodeName).stream())
            .collect(Collectors.groupingBy(classifier, () -> new LinkedHashMap<>(), collector)).values();
    //linkedhashmap ensures ordering
}
```

Ну и несколько утилок для удобства:

Утилки

```
/**
 * Same as above, but uses fixed replace rule to merge meta
 *
 * @param nodeName
 * @param classifier
 * @param <K>
 * @return
 */
public <K> Collection<Meta> collectNodes(String nodeName, Function<? super Meta, ? extends K> classifier) {
    return collectNodes(nodeName, classifier, MergeRule.replace());
}

/**
 * Same as above but uses fixed meta value with given key as identity
 * @param nodeName
 * @param key
 * @return
 */
public Collection<Meta> collectNodes(String nodeName, String key) {
    return collectNodes(nodeName, meta -> getValue(key, Value.NULL));
}
```