Java Programming
Lab Streams 3

Reducers

1. Lab objectives

The purpose of this lab is to get you familiar with some of the terminal methods

2. Lab Setup

You can do this whole lab in one project with different packages for the different parts

3. Lab Solutions

Solutions for each lab are provided in the repository in the labs directory for each module. These are not the only possible solutions and your solution to the labs may be a different in some ways than the reference solutions.

In order to make lab more challenging, the requirements are going to be stated and you will be responsible for coming up with a solution. There is a solution for each part provided but these are not necessarily the only solutions

Part 1: Reducers

1. Start with a list of Doubles as shown

```
List<Double> vals = Arrays.asList(4.35, 4.9, 123.0, 1.8);
```

2. Using the reduce() terminal method, calculate the sum of the numbers.
3. With the list of Integers, shown below, find the maximum value. Remember that you have to you use an optional as shown here. Consult the code in demo5b if you need some guidance.

```
List<Integer> data = Arrays.asList(3,2,9,4,3,9,4,1,9,0,3,5,4,2,7,6);
Comparator<Integer> comparator = Comparator.comparing(Integer::intValue);
```

4. Using allMatch() and anyMatch(), determine if there are any numbers that are divisible by 3 and if all of the numbers are less that 10. Kind of trivial, but it's about getting you used to writing the code. Consult demo5c if you need a hint.

Part 2: Collectors

1. Create a class called StockItem as shown below.

```
class StockItem {
    private int stockNumber;
    private String description;

    public StockItem(int stockNumber, String description, int unitPrice) {
        super();
        this.stockNumber = stockNumber;
        this.description = description;
    }
    public String getDescription() {
        return description;
    }
    public int getStockNumber() {
        return stockNumber;
    }

}
```

2. Create some dummy data making sure that all of the stock numbers are distinct, such as below:

```
    List<StockItem> inventory = Arrays.asList(
            new StockItem(1111,"Hi Resolution Monitor"),
            new StockItem(2222, "Ergonomic Mouse"),
            new StockItem(3333, "GPU Card"));
}
```

3. Use a collector to create a map of the form where the key value is the stockNumber and the value is the description.
4. Once your code is running, add a duplicate item as shown below and rerun the code. Why doesn't it work?

```
List<StockItem> inventory = Arrays.asList(
        new StockItem(1111,"Hi Resolution Monitor"),
        new StockItem(2222, "Ergonomic Mouse"),
        new StockItem(2222, "Standard Mouse"),
        new StockItem(3333, "GPU Card"));
```

5. Since you don't know in advance if any of the stock numbers are duplicated, what's an alternative approach? Why won't filter() work? How about a list of pairs of the form List<Integer,String>? What is the real challenge of implementing any approach that requires we filter out unique objects based on some value?

Part 3: Data transformations

Using the same data from the previous section, convert the stream of objects into a stream of strings in a Json format. Like this

```
StockItem(3333, "GPU Card) → "{'stockNumber' : '3333', 'description' : 'GPU Card'} "
```

This is an example of where we have to plan carefully to decide where to keep the code. We will have a class discussion on this before you check the answer.

End of Lab