1. Traders execute transactions.

```
Trader

private final String name;
private final String city;

public Trader(String n, String c);

public String getName();

public String getCity();

public String toString();
```

```
Transactions
private final Trader trader;
private final int year;
private final int value;
public Transaction(Trader trader, int year, int value);
public Trader getTrader();
public int getYear();
public int getValue();
public String toString();
```

You're asked by your manager to find answers to eight queries.

- Find all transactions in the year 2013 and sort them by value (small to high).
- What are all the unique cities where the traders work?
- Find all traders from Kolkata and sort them by name.
- Return a string of all traders' names sorted alphabetically.
- Are any traders based in Mumbai?
- Print all transactions' values from the traders living in Delhi.
- What's the highest value of all the transactions?
- Find the transaction with the smallest value.

2. Given a Stream where each element is a word, count the number of times each word appears. So, if you were given the following input, you would return a Map of [Ishan → 3, Paul → 2, Biswadip → 1]:
Stream<String> names = Stream.of("Ishan", "Paul", " Biswadip", " Ishan ", "Paul", " Ishan");

3.
```
public static int multiplyThrough(List<Integer> linkedListOfNumbers) {
    return linkedListOfNumbers.stream().reduce(5, (acc, x) -> x * acc);
```

The code multiplies every number in a list together and multiplies the result by 5. This works fine sequentially, but has a bug when running in parallel.
Make the code run in parallel using streams and fix the bug.

4. ***Artist***

An individual or group who creates music

- *name*: The name of the artist (individual or group name)
- *members*: A set of other artists who comprise this group - this field might be empty
- *origin*: The primary location of origin of the group (e.g., "Kolkata").

(a) Convert this code sample from using external iteration to internal iteration:

```
int totalMembers = 0;
for (Artist artist : artists) {
    if(artist.getOrigin()==”Kolkata”) {
            Stream<Artist>  members = artist.getMembers();
            totalMembers += members.count();
            }
}
```

(b) Find the artist with the longest name. You may use a Collector and the reduce higher-order function.