Lab 9 – Part I

1. In the code folder for this lab, there is a version of the startup code for the Swing project, and in the business package there is a `Main` class containing a `main` method. The `main` method calls several other methods; each of these attempts to extract information from the Library System. Implement these methods using stream pipelines (the body of each method should contain a *single line of code* – namely, the stream pipeline).

2. In the code folder for this problem, there are classes `Order` and `OrderItem`, along with a `Main` class that has a `main` method. The `main` method loads test data, populating a list of `Order` instances, each of which contains a list of `OrderItem` objects. It then calls `displayAllOrders`, which displays this test data to the console using formatted output; each `Order`, together with its `OrderItems`, is shown. The last method call in the `main` method is to an unimplemented method `showAllOrderItems`. This method is supposed to display all `OrderItems` separately, apart from the owning `Orders`. Carry out this implementation (using the technique described in the slides) by embedding the `Orders` list in a `Stream` and using `flatMap`.

3. In the code for this lab (for prob3) there is an `Employee` class. The `main` method of that class creates a list of `Employee` objects for testing purposes. Finish coding the main method by writing code to sort the list. Sorting order should be done first by name in ascending order, then by salary in descending order. Startup code is provided in your code folder.

4. This exercise asks you to work with potentially infinite streams of prime numbers.
   A. To begin, create a `final` variable `Stream<Integer> primes` that contains all prime numbers (in particular, the `Stream` is infinite). Generate the primes using the `iterate` method of `Stream` – do *not* use the `map` or `filter` `Stream` operations.

   B. Next, create a variation of the `primes` `Stream` that can be called multiple times by a method `printFirstNPrimes(long n)`, which prints to the console the first `n` prime numbers. Note that the `Stream primes` that you created in part A cannot be used a second time; how can you get around that limitation? Prove that you succeeded by calling the method `printFirstNPrimes(long n)` (from a `main` method) more than once.

   If you succeed, you should be able to run the following code without getting a runtime exception:

```
public static void main(String[] args) {
    PrimeStream ps = new PrimeStream(); //PrimeStream is enclosing class
    ps.printFirstNPrimes(10);
    System.out.println("====");
    ps.printFirstNPrimes(5);
}
```

5. In the startup code for this problem, you will find a class `Customer` and another class `Problem` that contains two static methods:

```
static List<String> elementsInBoth(List<String> list1, List<String> list2)
static List<String> getZipsOfSpecialCustomers(List<Customer> list)
```

You need to implement these methods using streams.

The method `elementsInBoth` returns a list of all `Strings` that occur in both of the two input lists. Below is an example of how this method should behave:

Example: If `list1 = {"A", "B", "D"}` and `list2 = {"B", "C", "D"}`, then the return list should be `{"B", "D"}` since both "B" and "D" occur in both lists.

The method `getZipsOfSpecialCustomers` returns a list of the zipcodes, in sorted order, of those Customers who live in a city for which the name of the city contains 6 or more characters, but which does not contain the letter 'e'. Your output list must not contain duplicate elements.

Example: Below are 5 customers.
Customer 1: ["Bob", "11 Adams", "Fairfield", "52556"]
Customer 2: ["Andy", "1000 Channing Ave", "Oskaloosa", "54672"]
Customer 3: ["Zeke", "212 Wilkshire Blvd", "Chicago", "57532"]
Customer 4: ["Tom", "211 Blake Ave", "Oskaloosa", "54672"]
Customer 5: ["Bill", "10 Wolfsen Blvd", "Orkin", "84447"]

When run on this customer list, the method should return the following list of zip codes:
["54672","57532"]
The Fairfield customer was ignored (because 'e' occurs in Fairfield) and the Orkin customer was ignored (because the length of "Orkin" is less than 6). Also, the multiple occurrences of an Oskaloosa zipcode were reduced to just one so that there were no duplicates in the final list. Note that the final zipcode list is in sorted order.

A `main` method has been provided that will help you test your implementations of both of these methods.

.