Lab 9

Day 1: do problems 1,2,3,5

Day 2: finish the rest of the problems

1. Short Answer:
   a. What happens when the following code is executed?

   ```
   public static void main(String[] args) {
      IntStream ones = IntStream.generate(() -> 1).distinct();
      ones.forEach(System.out::println);
   }
   ```

   Explain. What would be a quick way to fix this?

   b. You have a `Stream` of `String`s called `stringStream` consisting of the values “Bill”, “Thomas”, and “Mary”. Write the one line of code necessary to print this stream to the console so that the output looks like this:

   Bill, Thomas, Mary

2. This exercise asks you to work with potentially infinite streams of prime numbers.
   A. To begin, create a `final` variable `Stream<BigInteger> primes` that contains all prime numbers (in particular, the `Stream` is infinite).

   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);
   }
   ```

3. In the lecture demo `lesson9.lecture.comparators2.EmployeeInfoBetter`, we showed how to use `Comparator.comparing` and `thenComparing` to create better, more readable, and more functional-style `Comparator`s. In the demo code, however, there is branching logic that could be replaced by a cleaner design:

   ```
   public void sort(List<Employee> emps, final SortMethod method) {
      if(method == SortMethod.BYNAME) {
         Collections.sort(emps,
   ```

```java
            Comparator.comparing(byName)
                      .thenComparing(bySalary));
    } else {
       Collections.sort(emps,
            Comparator.comparing(bySalary)
                      .thenComparing(byName));
    }
}
```

Eliminate the branching logic by defining a `HashMap`. Start with the `EmployeeInfoBetter` and `Employee` classes from `lesson9.lecture.comparators2`, and then modify `EmployeeInfo` in a clever way that eliminates branching logic.

4. Implement a method

   ```java
   public static void printSquares(int num)
   ```

   which creates an `IntStream` using the `iterate` method. The method prints to the console the first `num` squares. For instance, if `num` = 4, then your method would output 1, 4, 9, 16. Note: You will need to come up with a function to be used in the second argument of `iterate`.

5. Create a method

   ```java
   Stream<String> streamSection(Stream<String> stream, int m, int n)
   ```

   which extracts a substream from the input stream `stream` consisting of all elements from position `m` to position `n`, inclusive; you must use only `Stream` operations to do this. You can assume `0 <= m <= n`. A Java class has been provided for you in the lab folder for this lesson; implement the method `streamSection` given in that class, and test using the `main` method provided.

6. Implement a method

   ```java
   public Set<String> union(List<Set<String>> sets)
   ```

   by creating a stream pipeline that transforms a list of sets (of type String) into the union of those sets. Make use of the `reduce` method for streams.

   Example: The union method should transform the list [{“A”, “B”}, {“D”}, {“1”, “3”, “5”}] to the set {“A”, “B”, “D”, “1”, “3”, “5”}.

7. In the package lesson9.labs.prob7a, there is an `Employee` class and a `Main` class, which has a `main` method that loads up a Stream of `Employee` instances.
   a. In the final line of the `main` method, write a stream pipeline (using filters and maps) which prints, *in sorted order (comma-separated, on a single line),* the full names (first name + “ “ + last name) of all `Employee`s in the list whose salary is greater than $100,000 and whose last name begins with any of the letters in the alphabet *past* the letter ‘M’ (so, any letters in the range ‘N’--‘Z’).

      For the main method provided in your lab folder, expected output is:

b. Turn your lambda/stream pipeline from part (a) into a Lambda Library element, following the steps in the slides. First, create a class LambdaLibrary; this class will contain only public static final lambda expressions. Then, identify the parameters that need to be passed in so that your lambda/stream pipeline can operate properly. Finally, think of a function-style interface (Function, BiFunction, TriFunction, etc) that can be used to accommodate your parameters and then name your pipeline, with the function-type interface as its type (as in the slide example). Call your Library element in the main method instead of creating the pipeline there, as you did in part (a).

8. In the package lesson9.labs.prob8, a Main class is provided that is essentially the same as the one used in Problem 7. Comments appear in the main method that indicate two queries that need to be executed. As in Problem 7, create a class LambdaLibrary that will store implementations of these queries as lambda pipeline expressions. Then call these expressions in the main method to verify they produce the expected results.

9. In the folder lesson9.labs.prob9 there are classes Book and BookCopy, as in the Library project. Use a lambda/stream pipeline to implement an isAvailable() method in Book that uses the stream operation reduce (Hint: a Book is available if copy1 is available OR copy2 is available OR…). To test your code, add a Main class to the package and run the following main method:

```java
public static void main(String[] args) {
    //set up
    Book book = new Book("test", 3);
    List<BookCopy> copies = book.getCopies();
    copies.forEach(copy -> copy.changeAvailability());

    //test
    System.out.println(book.isAvailable());
}
```