Recitation 13

Streams

The objective of this set of exercises is to write code to process data using streams and stream operations, not loops or recursion.

---

1. Write a method that given an array of integers, prints all odd numbers in the array in sorted order.

```
public static void sortedOdds(int[] arr) {
  // COMPLETE THIS METHOD
}
```

SOLUTION

```
public static void sortedOdds(int[] arr) {
    Arrays.stream(arr)
        .filter(i -> i % 2 != 0)
        .sorted()
        .forEach(System.out::println);
}
```

---

2. Write a method to partition the integers from 2 to some parameter n into primes and non-primes. You may implement support methods as needed.

```
public static Map<Boolean, List<Integer>>
primesAndNot(int n) {
  // COMPLETE THIS METHOD
}
```

SOLUTION

```
public static boolean isPrime(int n) {
  int sqrtn = (int)Math.sqrt(n);
  return
    IntStream.rangeClosed(2,sqrtn)
          .noneMatch(i -> n % i == 0);
}

public static Map<Boolean, List<Integer>>
primesAndNot(int n) {
  return
```

```
    IntStream.rangeClosed(2,n).boxed()
          .collect(partitioningBy(i -> isPrime(i)));
```

---

3. Write a method to return the common items in two integer lists:

```
public static int[] getCommon(List<Integer> l1, List<Integer> l2) {
  // COMPLETE THIS METHOD
}
```

SOLUTION

```
public static Integer[] getCommon(List<Integer> l1, List<Integer> l2) {
  return
    l1.stream()
     .flatMap(i ->
              l2.stream()
               .filter(j -> i == j))
     .toArray(Integer[]::new)
}
```

Alternatively, we can have the method return an int[] but to do this, we will need to map the stream to an IntStream, then apply toArray

```
public static int[] getCommon(List<Integer> l1, List<Integer> l2) {
  return
    l1.stream()
     .flatMap(i ->
              l2.stream()
               .filter(j -> i == j))
     .mapToInt(i -> i)
     .toArray()
}
```

---

4. Pythagorean triples are any three positive integers a, b, c for which $a^2 + b^2 = c^2$

Write a method to print all Pythagorean triples for which a and b are within a given limit:

```
// prints all pythagorean triples a,b,c where a and b are <= limit
// each triple should be printed as "a,b,c"
public static void printPythagoreans(int limit) {
  // COMPLETE THIS METHOD
}
```

SOLUTION

```
public static void printPythagoreans(int limit) {
  IntStream
   .rangeClosed(1,limit)
   .boxed()
   .flatMap(a ->
         IntStream
          .rangeClosed(a, limit)
          .boxed()
          .filter(b -> Math.sqrt(a*a + b*b) % 1 == 0)
          .map(b -> new int[] {a, b, (int)Math.sqrt(a*a+b*b)}))
   .forEach(t -> System.out.println(t[0] + "," + t[1] + "," + t[2]);
```

---

5. Write a method to generate fibonacci numbers upto a limit. The first two fibonacci numbers are 0 and 1. So if the limit is 10, the numbers generated will be [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] (Hint: use a 2-item array to store a pair of consecutive fibonacci numbers, and iterate on such 2-item arrays to do the progression)

```
public static int[] fibs(int limit) {
  // COMPLETE THIS METHOD
}
```

SOLUTION

```
public static int[] fibs(int limit) {
  return
    Stream.iterate(new int[]{0,1},
              f -> new int[]{f[1],f[0]+f[1]})
        .limit(limit)
        .mapToInt(f -> f[0])
        .toArray();
}
```

---

6. Write a method to classify words in a file according to length:

```
public static Map<Integer,List<String>> wordsByLength(String file)
throws IOException {
  // COMPLETE THIS METHOD
}
```

SOLUTION

```
public static Map<Integer,List<String>> wordsByLength(String file)
throws IOException {
  return
    Files.lines(Paths.get(file))
        .map(l -> l.split(" "))
        .flatMap(Arrays::stream)
        .distinct()
        .collect(groupingBy(String::length));
}
```

---

7. There are several parts to this question, which are all about hackathons.

   A hackathon tracking site keeps info on all school hackathons as follows:

```
public class Hackathon {
  private String school;
  private int year;
  private int month;   // 1..12
  private int points;      // awarded by site, out of 1000
  private int participants;
  private int sponsorship;  // $$ value

  // assume getter methods for the instance fields
  // i.e. getSchool(), getYear(), etc.
  ...

  public String toString() { return school; }
}
```

   A school may hold more than one hackathon in a year (Rutgers USACS holds two per year).

   Assume that all hackathons are stored in a List<Hackathon> instance called hackathons.

   For each of the following questions, use streams to get the required result. Assign the result to a typed and named variable.

   You may assume that all the required classes and methods have been imported so you can use them without qualifiers.

   Some of the questions may have alternative answers to the ones given here - with streams there is often more than one way to solve a problem.

   1. How many hackathons were held in 2015?

      SOLUTION

```
int h2015 = (int)
hackathons.stream()
        .filter(h -> h.getYear() == 2015)
        .count();
```

2. What was the highest sponsorship amount for any hackathon?

   SOLUTION

```
int maxSponsorship =
hackathons.stream()
        .map(h -> h.getSponshorship())
        .reduce(Integer::max)
        .orElse(0);
```

3. Which school (name any one) had the largest number of participants in any hackathon in 2015?

   SOLUTION

```
String school =
hackathons.stream()
        .filter(h -> h.getYear() == n)
        .collect(maxBy(comparing(Hackathon::getParticipants)))
        .map(Hackathon::getSchool)
        .orElse("No school");
```

4. For each year, get the number of schools that hosted hackathons with > 500 participants in that year

   SOLUTION

```
Map<Integer,Long> numSchools =
hackathons.stream()
        .filter(h -> h.getParticipants() > 500)
        .collect(groupingBy(Hackathon::getYear, counting()));
```

5. Modify the result of the previous problem to name all the schools instead of the count, sorted in decreasing order of participants. (A school should appear only once in a year even if it hosted multiple hackathons that year that had > 500 participants.)

   SOLUTION

```
Map<Integer,Set<String>> numSchools =
hackathons.stream()
        .filter(h -> h.getParticipants() > 500)
```

```
        .sorted(comparing(Hackathon::getParticipants).reversed())
        .collect(groupingBy(Hackathon::getYear, mapping(Hackathon::getSchool,toSet())));
```