**Fully Functional**

Below are some more lambda expression exercises / advanced nonsense, now that you are a little more comfortable with streams / lambdas / the Java 8 APIs.

1. The java.util.Arrays.toList method will return some number of comma-separated elements (known as "var args") as a List; very useful for test purposes. Examples:
   1. List<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3));
   2. List<String> list = new ArrayList<>(Arrays.asList("hello", "and", "goodbye"));

**Note:** The value Arrays.toList returns must be passed to ArrayList's constructor as shown above, as it returns an immutable list and couldn't be changed if saved directly into a List variable.

1. You don't need to read the "Functional Interfaces" presentation yet. Nice! Get three names from Standard In (from the console / user) and collect them into upper case into a List<String>. A silly thing to do, but useful to review from the previous lab.
   1. Make a regular old Scanner to get input from System.in.
   2. Stream.of - creates a stream given any number of comma-separated values (var args).
2. Create a single String of all the elements in a List<String>, separated by a comma and a space.
   1. Collectors.joining("separator") - A utility method (like Collectors.toList) that returns a Collector object, for converting a stream into a collection.
3. Sort an Integer[] in descending order.
   1. Arrays.sort - Has an overloaded method that accepts the array and a Comparator, which you've already seen can be done in-line with a lambda.
4. In one line, print "Hello!" n times (the one line does not count the declaration/initialization of n).
   1. IntStream.range - Returns a sequential stream of integers from start (inclusive) to finish (exclusive).
5. In one line, print the contents of a file (on separate lines) stored in the current working directory to the console.
   1. Files.lines - Returns a stream of all the lines in the file, as Strings.
   2. Paths.get - Returns a path to a file given the supplied String.
6. In one line, determine if a number is prime or composite. Check IntStream's methods for this.
7. In one line, generate a List<Integer> comprised of m random numbers from 1-100 (inclusive).
   1. new Random().ints(start, end) - Returns an infinite stream of pseudo-random integers from start to end (inclusive, exclusive). Called on a Random object.
   2. limit(size) - Returns a stream truncated to size elements.
   3. boxed - Returns the int stream as an Integer stream (useful for using with collections).
8. Copy your code from the previous problem and modify it to only include *unique* randoms.
   1. distinct - Returns an IntStream of only the distinct elements in the calling stream.
9. Sort an int[] in descending order.
   1. Arrays.stream(array) - creates a stream of integers from the supplied array.
   2. The Arrays.sort(array, Comparator) method you've seen only works with Integer objects, you'll need to box the primitives before sorting like you did previously.
   3. toArray - called on an IntStream, will convert the stream into an array of primitives.
10. In one line, create (and start) a new Thread that forever prints "Thread running".
    1. Thread's constructor expects a Runnable, something that implements the Runnable interface (and overrides its single abstract method void run()).
       1. Lamba syntax is convenient for this! Recall that if a lambda expression "body" contains more than one statement, they must be grouped with curly brackets.
11. Complete the "Repeated String" problem found in the lab folder (from of hackerrank.com). Test it on HackerRank (search by problem name, you can make a login or use a Google login).
    1. Use lambda expressions to count the number of 'a' characters in one line (rather than using a for loop or regex).
       1. chars - returns an IntStream of all the ASCII values of the characters in the String.
12. Complete this [CodingBat](https://codingbat.com/prob/p165312) problem in one line.
    1. mapToObj - perform a mapping operation, given the supplied function. Called on an IntStream.
13. Read the "Functional Interfaces" presentation if you haven't yet.

Write a method static int sumIf(List<Integer> numbers, Predicate<Integer> condition) that will sum the values in list given the supplied condition (i.e. if they meet this condition).

* 1. Test (using lambda expressions for condition) with a list of numbers using various conditions, e.g. only even numbers or numbers below a particular value.

1. **(Advanced)** Sort a HashMap based on its values.
   1. entrySet - returns a Set<Entry>, where Entry is a static nested class in Map, basically a bundle of a map entry's key/value pair. Can be streamed.
   2. sorted - returns a sorted stream of Entry objects, given the supplied Comparator.
   3. Collectors.toMap - Returns a Collector that accumulates elements into a Map whose keys and values are the result of applying the provided mapping functions to the input elements.
      1. Use Map.Entry::getKey and Map.Entry::getValue to use an Entry's key and value to perform the mapping operation.
      2. **Note:** map the results to a LinkedHashMap instance, which maintains the order of the inserted elements. Using a HashMap again it will appear unsorted.

Here's some code for testing, no charge:

Map<String, Integer> map = new HashMap<>();

map.put("Computer", 2\_000);

map.put("Sandwich", 5);

map.put("Coffee", 2);

map.put("Car", 30\_000);

map.put("Pencil", 1);