Sreeja Rao

S561636

1. What is wrong in below code?

Character[] chars1 = {'D','B','A','C'}; Stream stream=Stream.of(chars1).sorted(); System.out.println(stream1.findFirst());System.out.println(stream1.skip(2).findFirst());

Replace Stream.of(chars1) with Arrays.stream(chars1) to make a stream from an array of characters.

In case we expected to utilize a few conclusion methods such as skip() and findFirst() on the identical stream, we must either reproduce the stream or store temporary outcomes. After an conclusion operation, streams are seen as depleted.

2. What is output of below code and explain it?

int[][] m = {{1,2},{3,4},{5,6}};

System.out.println(Stream.of(m) .mapToInt( e-> IntStream.of(e).sum()) .sum());

Output of the code is 21.

The code to begin with constructs a 2D array called m. It at that point employments streams to calculate the overall of each row within the sub-array, and eventually includes up all of those sums.

3. Given String[] n= {"John","Peter","Susan","Kim","kim"};

Write the code using streams to print the first element in stream in lowercased.

A screenshot of a computer code

Description automatically generated

4. What is output of below code?

Given String[] n= {"John","Peter","Susan","Kim","kim"};

Stream.of(n).mapToInt(e -> e.length()).sum());

The code adds up all of the string lengths within the array n. In any case, since there isn't a System.out.println() instruction to show the sum, the output will be blank.

5. Given values = {2,7,8,3,10,20,15,10,8}. The average is 12.66

a. Write traditional Java code to print the average of distinct even numbers greater than 2.

A screenshot of a computer code

Description automatically generated

b. write Java code to print average of distinct even numbers greater than 2 using streams.

A screenshot of a computer program

Description automatically generated

6. Explain in detail what did you observe after executing below Code 1 and Code 2?

String[] names={"John","Peter","Susan","Kim","Jen"};

Set set1 = new HashSet<>(Arrays.asList(names));

Code 1:

long startTime = System.currentTimeMillis();

Object[] s= set1.parallelStream() .filter(e -> e.length()>3) .sorted().toArray(); System.out.println(java.util.Arrays.toString(s));

long endTime = System.currentTimeMillis(); System.out.println("parallel execution filter first Sorted later is "+(endTime-startTime)+" milliseconds");

Code 2:

long startTime2 = System.currentTimeMillis();

Object[] s1= set1.parallelStream() .sorted().filter(e -> e.length()>3) .toArray(); System.out.println(java.util.Arrays.toString(s1));

long endTime2 = System.currentTimeMillis(); System.out.println("parallel execution Sorted first Filter later is "+(endTime2-startTime2)+" milliseconds");

The most difference between Code 1 and Code 2 is the order in which the operations are performed. They simultaneously sort and filter names, but since HashSets are unordered, the comes about of these operations may change depending on the sequence they are performed (sort first, then filter vs. filter first).