Streams assignments

**Setup:**

Create the following classes:

class Fruit { String name;int calories;int price; String color; }

class News {intnewsId; StringpostedByUser;String commentByUser;String comment; }

class Trader { String name; String city; }

class Transaction { Trader trader; int year; int value; }

1. Display the fruit names of low calories fruits i.e. calories < 100 sorted in descending order of calories.

**fruits.stream().**

**filter((Fruit fruit)->fruit.calories< 100).**

**sorted(Comparator.comparing(Fruit::getCalories).reversed()).**

**forEach(System.out::println);**

1. Display colorwise list of fruits.

Map<String, List<Fruit>>mapFruitColor =

fruits.stream().

collect(groupingBy(Fruit::getColor));

System.out.println("\n\n\nColorwise fruits: " + mapFruitColor + "\n\n\n");

1. Display only RED color fruits sorted as per their price in ascending order.

fruits.stream().

filter((Fruit fruit)->fruit.getColor().equalsIgnoreCase("Red")).

sorted(Comparator.comparing(Fruit::getPrice)).

forEach(System.out::println);

1. Find out how many times the word 'budget' arrived in user comments of all news. Assume that ‘budget’ word has arrived maximum once in a given user comment.

intnewsCountForWordBudget =

newsList.stream().

mapToInt((News news)-> {

inti = 0;

Pattern p = Pattern.compile("budget");

Matcher m = p.matcher(news.getComment() );

while (m.find()) {

i++;

}

returni;

}).

sum();

System.out.println("\n\n\nWord 'budget' has appeared " + newsCountForWordBudget + " times in comments");

1. Find all transactions in the year 2011 and sort them by value (small to high).

List<Transaction>filteredTransactions =

transactions.stream().

filter((Transaction transaction)->transaction.getYear() == 2011).

sorted(Comparator.comparing(Transaction::getValue)).

collect(toList());

System.out.println("\n\n\nTransactions in the year 2011 sorted by their value: " + filteredTransactions);

1. What are all the unique cities where the traders work?

List<String>traderCities =

transactions.stream().

map((Transaction transaction)->transaction.getTrader().getCity()).

distinct().

collect(toList());

System.out.println("\n\n\nTraders work in cities: " + traderCities);

1. Find all traders from Pune and sort them by name.

List<Trader>puneTraders =

transactions.stream().

filter((Transaction transaction)->transaction.getTrader().getCity().equals("Pune")).

sorted((Transaction t1, Transaction t2)->t1.getTrader().getName().compareTo(t2.getTrader().getName())).

map((Transaction transaction)->transaction.getTrader()).

collect(toList());

System.out.println("\n\n\nTraders in Pune: " + puneTraders);

1. Return a string of all traders’ names sorted alphabetically.

String traderNames =

transactions.stream().

map((Transaction transaction)->transaction.getTrader().getName()).

sorted().

collect(joining(","));

System.out.println("\n\n\nTrader names: " + traderNames);

1. Are any traders based in Indore?

Optional<Transaction>opTransaction =

transactions.stream().

filter((Transaction transaction)->transaction.getTrader().getCity().equals("Indore")).

findAny();

System.out.println("\n\n\nAre any traders based in Indore? " + opTransaction.isPresent());

1. Print all transactions’ values from the traders living in Delhi.

List<Integer>transactionValues =

transactions.stream().

filter((Transaction transaction)->transaction.getTrader().getCity().equals("Delhi")).

map((Transaction transaction)->transaction.getValue()).

collect(toList());

System.out.println("\n\n\nTransaction values from the traders living in Delhi: " + transactionValues);

1. What’s the highest value of all the transactions?

Optional<Integer>opMaxTransactionValue =

transactions.stream().

map((Transaction transaction)->transaction.getValue()).

reduce(Integer::max);

System.out.println("\n\n\nMax transaction value: " + opMaxTransactionValue.get());

1. Find the transaction with the smallest value.

Optional<Integer>opMinTransactionValue =

transactions.stream().

map((Transaction transaction)->transaction.getValue()).

reduce(Integer::min);

System.out.println("\n\n\nMin transaction value: " + opMinTransactionValue.get());

1. Print first 15 prime numbers.

Stream.iterate(2, (n)->n+1).

filter((n)-> {

for(int i=2;i<n;i++) {

if(n%i==0)

return false;

}

return true;

}).

limit(15).

forEach(System.out::println);

Grouping & Partitioning

1. Print all high calories Fruits(Calorie > 200) using partitioning.

fruitList.stream().

collect(Collectors.partitioningBy((Fruit fruit)-> fruit.calories > 200)).

get(true).stream().

forEach(System.out::println);

1. Find out the newsId which has received maximum comments.

Entry<Integer, Long>mapNewsComments =

newsList.stream().

collect(groupingBy(News::getNewsId, counting())).

entrySet().stream().

max(Comparator.comparing(Entry::getValue)).

get();

System.out.println("\n\n\nNews Id: " + mapNewsComments.getKey() + " has maximum number of comments i.e. " + mapNewsComments.getValue());

1. Display commentByUser wise number of comments.

Map<String, Long>mapCommentByUser =

newsList.stream().

collect(groupingBy(News::getCommentByUser, counting()));

System.out.println("\n\n\ncommentByUser wise number of comments: " + mapCommentByUser);

1. Find out which user has posted maximum comments.

Entry<String, Long>mapUserComments =

newsList.stream().

collect(groupingBy(News::getCommentByUser, counting())).

entrySet().stream().

max(Comparator.comparing(Entry::getValue)).

get();

System.out.println("\n\n\nUser: " + mapUserComments.getKey() + " has maximum number of comments i.e. " + mapUserComments.getValue());