CS2230 Computer Science II: Data Structures
Homework 7
Better Abstractions for Queries

Table of Contents

1 Goals for this assignment

- Generalize common patterns for working with Lists, Sets, and Maps
- Use Sets and Maps to write efficient algorithms
- Use a custom class as a key in a TreeMap and HashMap
- Debug programs using JUnit tests

2 Introduction

In Homework 6, you wrote code to answer queries about a flight database. In this homework, you will ***refactor*** (i.e., rewrite) your queries to improve the code's reusability and extensibility.

2.1 Improvement #1: Declarative queries

As you implemented more queries, you may have noticed some common operations. Examples include:

- Filter – only keep records whose attributes have certain values, e.g., "find flights with an ORIGIN of CID"
- Transform – convert records or other elements to another format, e.g., "get the ORIGIN of every flight"
- Distinct – only keep one copy of each occurring element, e.g., "find the distinct ORIGINs of the flights"
- Count – count how many records or elements there are, e.g., "find the number of flights…"

As you implement these common operations, you will rewrite your queries to take advantage of them. In fact, you will end up eliminating all if-statements and loops from your queries! This idea of a collection of reusable, common operations for writing queries is called a ***declarative query language***. Declarative query languages are the "programming language of choice" for most real database management systems. When designed well, a declarative language lets the programmer focus more on WHAT to do to the data than on HOW to do it.

2.2 Improvement #2: Cacheable queries

In general, queries can take a long time to run. To speed up repeated executions of a query, we can use ***caching***. Caching is the idea of storing the results of a query so that next time it is asked, we can just return the result instead of re-computing it.

3 Setup

1. Clone from https://research-git.uiowa.edu/cs2230-assignments/hw7-flights-sp21.git
2. Push the initial code to your private repository: https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

3. Set up an IntelliJ project with the hw7-flights-refactored-sp21 folder.
4. Download commons-csv-1.8.jar from ICON. Put commons-csv-1.8.jar into your hw7-flights-refactored-sp21 folder.
5. Download flights.zip from ICON. Put flights.zip into your hw7-flights-refactored-sp21 folder. Then unzip it. The result is that flights1990.csv, flights2005.csv, and flights2020.csv are in hw7-flights-refactored-sp21 folder.
6. You should now be able to run the tests in QueryTest.java.

Just an FYI: Why didn't we include the jar and csv files in the git repository?

- Typically, a git repository is only meant for source code. It is not a good place to put data (e.g., the large csv files) or external libraries (e.g., the jar file). We call such files "dependencies", and they should be distributed in a separate way from the source code.
- We have set things up so that you do not accidentally commit the jar or csv files. To do so, we've included a .gitignore file that tells git what types of files never to commit. You do not need to modify that file.

4 Overall approach to this homework

There are three main kinds of tasks:

a) Implement common operations (e.g., Transform) (section 5 - 14)
b) Rewrite queries (e.g., Query1) in terms of the common operations (section 5 - 14)

The queries (b) are dependent on the operations (a). That means you'll usually need to implement the operations (a) from previous sections to get the query (b) to work. Fortunately, all of the queries (b) themselves are independent of each other. That means you can certainly skip queries (b) you are having difficulty with.

c) Make caching work (section 15), by modifying AbstractQuery

The caching (c) is completely independent of finishing operations (a) and queries (b). You can work on it whenever you want.

5 (The new) structure of a query

You will be rewriting Query1-8 and Query10 from Homework 6. We've changed some aspects of the QueryX classes for the purposes of this homework.

A Query is run in two steps. First, you create it by calling its constructor on a filename (a String) saying where to get the input data from. Second, you call execute to actually calculate run it.

The query for Query0 is

How many flights were there from Cedar Rapids (CID) to Chicago (ORD) in the month of August (8)?

You can find the implementation of this query in Query0.java. In Homework 6, this query used a for-loop and if-statement. Now it just calls two operations: Filter.filter and Count.count.

5.1 Filter

The Filter.filter (defined in Filter.java) operation takes in an Iterable<FlightRecord> and returns an Iterable<FlightRecord> containing only those records that meet a particular condition. This condition is provided as a Function<FlightRecord, Boolean>.

So what is a Function<T,R>? It is a Java interface from the standard library. It has a single method ***apply()***, with one argument of type T and a return value of type R. So, classes that implement Function<T,R> need to provide an implementation of the apply() method.

So how does Filter.filter work? See the comments and code in Filter.java. By using a Function as the condition, instead of a specific condition like r.DEST.equals("DSM"), we can use Filter.filter with any condition!

To use the Filter.filter, Query0 just needs to provide the input and a Function<FlightRecord,Boolean> whose apply method checks the desired condition. This Function is the class IsLaxToOrdInAugust.

5.2 Count

The Count.count (defined in Count.java) operation takes in an Iterable<E> and returns the number of elements in it. This one is more self-explanatory; see Count.java if you are interested in the implementation.

5.3 So what?

So, what did this accomplish? Are we just hiding the for-loops and if-statements? Well, that is certainly part of it. But the even bigger improvement is that you'll be able to use Filter and Count in other queries, too! This will reduce the amount of duplicated code between queries.

6 Query 1

How many flights out of Cedar Rapids (CID) were there in total? Return your result as a single integer.

Rewrite this query in terms of Filter and Count. Keep the following in mind:

- The execute() method should not contain for-loops or if-statements
- You will need to write a custom class for use with Filter. Make it a private static class inside of Query1.java, as done with Query0's IsLaxToOrdInAugust.
- The execute() method has zero arguments. The input is provided as an instance variable called "input". You ***do not*** need to initialize that variable; initialization is done for you (by AbstractQuery). See Query0 for how it uses input.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

7 Query 2

What are the distinct destinations that can be reached in one flight from Cedar Rapids (CID)? Return each result as a String with the format DEST, DEST_STATE_ABR, for example, "HPN, NY".

This query is going to need two new operations: Transform and Distinct.

7.1 Transform

The Transform.transform (defined in Transform.java) operation takes an Iterable and returns another Iterable where each element is "transformed" (i.e., changed) in some way. Actually, Transform.transform takes another argument: a Function<To,From>. This Function is used to change each element.

See TransformTest.java for examples of how Transform.transform is used. For example, in testStringLength() we transform each String into an Integer that represents its length. To make Transform.transform do that for us, we give it a StringLength object. The StringLength class is defined in elsewhere in TransformTest.java. As you can see, its apply() method takes in a String and returns an Integer, the length of the String. Take a look at the other tests in TransformTest.java, too, to see how Transform.transform can be used with different Functions!

Complete Transform.transform in Transform.java. You should look at Filter.filter for the most similar example; except, you'll use the return value of apply() differently! Debug until you pass the tests in TransformTest.java.

7.2 Distinct

The Distinct.distinct operation (defined in Distinct.java) takes in an Iterable and returns a Set containing that Iterable's elements, thereby eliminating duplicates.

Complete Distinct.distinct in Distinct.java. Debug until you pass the tests in DistinctTest.java.

7.3 Rewrite Query 2

Now you can write Query2 in terms of Filter, Transform, and Distinct.

- The execute() method should not contain for-loops, if-statements, or call the methods of any data structures
- You will need to write custom classes for use with Filter and Transform
- HINT: use Transform to extract a String from each FlightRecord

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

8 Query 3

How many distinct destinations can be reached in one flight from Cedar Rapids (CID)? Return your result as a single integer.

Rewrite Query3 in terms of only the operations defined so far (Filter, Count, Transform, and/or Distinct). The execute() method should not contain for-loops, if-statements, or call the methods of any data structures.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

9 Query 4

How many flights out of Cedar Rapids (CID) were there for each destination? You only need to include destinations for which there was at least 1 flight from CID. Return each result as a String with the format DEST=number of flights. For example, if there were 10 flights from CID to LAX, then “LAX=10”.

This query is going to need one new operation: CountBy

9.1 CountBy

The CountBy.countBy operation (defined in CountBy.java) takes in an Iterable<E> and returns a Map<E, Integer>, where the keys are the elements of the Iterable and the values are the number of times a key appeared in the Iterable.

Complete CountBy.countBy in CountBy.java. Debug until you pass the tests in CountByTest.java.

9.2 Rewrite Query 4

Rewrite Query4 in terms of the common operations defined so far (Filter, Count, Transform, Distinct, and/or CountBy). The execute() method should not contain for-loops, if-statements, or call the methods of any data structures.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

10 Query 5

What month had the most flights (and how many flights was that)? Return your result as MONTH had NUM flights. For example, if the month was 7 and there were 500 flights then "7 had 500 flights".

This query is going to need one new operation: MaxOfMap

10.1 MaxOfMap

The MaxOfMap.maxOfMap operation (defined in MaxOfMap.java) takes in a Map<K,Integer> and returns a Map.Entry<K,Integer>, specifically the entry of the map whose value is the largest.

Tip: to get the entries of a Map<K,V> in the form of a Set<Map.Entry<K,V>>, you can call Map's entrySet() method.

Complete MaxOfMap.maxOfMap in MaxOfMap.java. Debug until you pass the tests in MaxOfMapTest.java.

10.2 Rewrite Query 5

Rewrite Query5 in terms of the common operations defined so far (Filter, Count, Transform, Distinct, CountBy, and/or MaxOfMap). The execute() method should not contain for-loops, if-statements, or call the methods of any data structures.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

11 Query 6

What pair of states has the most flights between them in either direction? Return your result as a single String with the two states in alphabetical order. For example, if the two states are IA and IL, then “IA, IL”.

Rewrite Query6 in terms of the common operations defined so far (Filter, Count, Transform, Distinct, CountBy, and/or MaxOfMap). The execute() method should not contain for-loops, if-statements, or call the methods of any data structures.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

12 Query 7

What distinct states can you NOT get to from an Iowa (IA) airport in one flight? Only include states that are present somewhere in the dataset (whether as an origin or destination).

This query is going to need one two new operations: Concatenate and Difference

12.1 Concatenate

The Concatenate.concatenate operation (defined in Concatenate.java) takes in two Iterables and returns an Iterable containing the elements of the first followed by the elements of the second. This method should not modify its inputs.
For example, if the Iterables are lists [4,5,6] and [3,6,8,9], then it returns [4,5,6,3,6,8,9].

Complete Concatenate.concatenate in Concatenate.java. Debug until you pass the tests in ConcatenateTest.java.

12.2 Difference

The Difference.difference operation (defined in Difference.java) takes in two Sets A and B and computes the set difference A \ B. That is, it returns a Set with all the elements of A except for those in B. This method should not modify its inputs.

For example {“a”,”b”} \ {“c”, “b”} results in {“a”}.

Complete Difference.difference in Difference.java. Debug until you pass the tests in DifferenceTest.java.

12.3 Rewrite Query 7

Rewrite Query7 in terms of the common operations defined so far (Filter, Count, Transform, Distinct, CountBy, MaxOfMap, Concatenate, and/or Difference). The execute() method should not contain for-loops, if-statements, or call the methods of any data structures.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

13 Query 8

For each state, what is the percentage of flights that stay within that state? For example, CID to DSM stays within Iowa. Return each result as STATE=PERCENT. For example, if the percentage of within-state flights in Florida is 5%, then "FL=0.050". Round and display the percentage with three digits. Here is example code to do that.

```
float y = 0.1234567
DecimalFormat df = new DecimalFormat("#.000");
String withThreeDigits = df.format(y);
```

This query is going to need one new operation: MapJoin

13.1 MapJoin

The MapJoin.mapJoin operation (defined in mapJoin.java) takes in two Maps, whose key types (K) are the same but whose value types (V1,V2) could be different. It returns a Map<K,Pair<V1,V2>>, where each Pair is formed by the values of the matching keys from the input Maps.

Tip: We recommend iterating over one of the Maps using either entrySet() or keyset().

Complete MapJoin.MapJoin in MapJoin.java. Debug until you pass the tests in MapJoinTest.java.

13.2 Rewrite Query 8

Rewrite Query8 in terms of the common operations defined so far (Filter, Count, Transform, Distinct, CountBy, MaxOfMap, Concatenate, Difference, and/or MapJoin). The execute() method should not contain for-loops, if-statements, or call the methods of any data structures.

HINT: use a MapJoin to get each numerator and denominator into a Pair together.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

14 Query 10 (that's right, there is no Query 9)

What distinct routes are there from CID to LAX that consist of exactly two flights (i.e., has a "layover")? Such a route exists if the destination of the first flight is the origin of the second flight. Return each result as a String with the format $ORIGIN_1$->$DEST_1$->$DEST_2$. For example, if there is a flight from CID to ALB and a flight from ALB to LAX, then "CID->ALB->LAX". We don't include $ORIGIN_2$ in the output because it is the same as $DEST_1$.

You are giving all "theoretical" routes. That is, we are ignoring the times of the first and second flight. The records don't include takeoff and landing times, so we can't know whether a passenger could really make the transfer from the first flight to the second flight.

This query will need two new operations: GroupBy and Flatten

14.1 GroupBy

The GroupBy.groupBy operation (defined in GroupBy.java) takes in an Iterable<E> and returns a Map<K, List<E>>. Where does the K come from? GroupBy also takes a Function<E, K>, which should be used to "extract" a key from an element. The List<E> mapped to by a key is the list of elements that have that key.

So basically, GroupBy is useful for taking a bunch of elements (such as FlightRecords) and organizing them by key, where the key is taken from the elements (such as one attribute of a FlightRecord).

Complete GroupBy.groupBy in GroupBy.java. Debug until you pass the tests in GroupByTest.java.

14.2 Flatten

The Flatten.flatten operation (defined in Flatten.java) takes an Iterable of Iterable<E> and returns an Iterable<E> containing all the elements. For example, if the Iterables were lists: [[10,20,30],[],[40],[50,60]] turns into [10,20,30,40,50,60].

Complete Flatten.flatten in Flatten.java. Debug until you pass the tests in FlattenTest.java.

14.3 Rewrite Query 10

Rewrite Query10 in terms of the common operations defined so far (Filter, Count, Transform, Distinct, CountBy, MaxOfMap, Concatenate, Difference, MapJoin, and/or GroupBy). The

execute() method should not contain for-loops*, if-statements, or call the methods of any data structures.

See (17. Query 10 hints) if you need hints.

*Although in this particular query, it is likely you will use a Transform whose Function uses a nested for-loop.

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

15 Caching

In general, queries can take a long time to run. To speed up repeated executions of a query, we can use ***caching***. Caching is the idea of storing the results of a query so that next time it is asked, we can just return the result instead of re-computing it.

15.1 QueryExecutor benchmarks

The QueryExecutor<R> class demonstrates this idea. As it runs queries, it stores the results in a Map<Query<R>, R>. This way, if it sees the same query again, it can just immediately return the results.

If you run the main() method, you'll see ***benchmarks*** (performance tests) run. You'll see a lot of runs, for the cases {first run, second run} X {1990, 2005, 2020} X {Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9} X {HashMap, TreeMap}. The most important thing to pay attention to is that every query is run twice. Right now, Query is not ready to be a key in a Map. Therefore, QueryExecutor never finds a Query, and so it runs every Query again. You'll see that every second run is no faster than the first run. For example…

```
HashMap cache
Query0 on flights1990.csv took 0.06707362 seconds (executed)
Query0 on flights1990.csv took 0.00795552 seconds (executed)
Query0 on flights2005.csv took 0.007279131 seconds (executed)
Query0 on flights2005.csv took 0.007734306 seconds (executed)
…
TreeMap cache
Query0 on flights1990.csv took 0.06707362 seconds (executed)
Query0 on flights1990.csv took 0.00795552 seconds (executed)
Query0 on flights2005.csv took 0.007279131 seconds (executed)
Query0 on flights2005.csv took 0.007734306 seconds (executed)
…
```

What your task is: make sure that Query can function properly as a Map key. You'll know you were successful when the cache appears to be doing its job (because it's finding the Query's it put into the Map). For example, the output will look more like…

HashMap cache
Query0 on flights1990.csv took 0.029566771 seconds (executed)
Query0 on flights1990.csv took **8.6235E-5 seconds** (**found in cache**)
Query0 on flights2005.csv took 0.012285444 seconds (executed)
Query0 on flights2005.csv took **5.8047E-5 seconds** (**found in cache**)
…
TreeMap cache
Query0 on flights1990.csv took 0.029566771 seconds (executed)
Query0 on flights1990.csv took **8.6235E-5 seconds** (**found in cache**)
Query0 on flights2005.csv took 0.012285444 seconds (executed)
Query0 on flights2005.csv took **5.8047E-5 seconds** (**found in cache**)

15.2 The details of making Query work as a key

There's no need to make changes to Query0, Query1, Query2, etc! That's because we've made all of them extend the AbstractQuery class. For this task, you should make all of your changes there. You need to make AbstractQuery work as a key for both HashMap and TreeMap.

- Equality: consider two AbstractQuery objects to be equal if their filenames and names are equals. ***Do not*** use the input Iterable in calculating equality. It is redundant with filename.
- Comparable: it doesn't matter what total ordering you pick for AbstractQuery. But it better be ***consistent with the equality*** of AbstractQuery objects.
- HINT: you had practice with this in Prelab/Lab 11

STOP! Before moving on, commit and push your code and check in your web browser that it is at https://research-git.uiowa.edu/cs2230-sp21/hw7-hawkid.

16 Optional reading

16.1 Simplifying the syntax of Functions

It can get understandably tedious to have to write so many classes that implement the Function interface, every time you use Transform, Filter, or GroupBy. Fortunately, Java supports syntactic sugar that simplifies this kind of code. The feature is called ***lambda expressions***. There are two examples in LambdaExample.java. Feel free to beautify your Queries with this feature.

16.2 More generalization

CountBy and GroupBy are named similarly for a reason. They both create Maps to organize the input elements by keys. We could make CountBy as a special case of GroupBy but only if we generalized GroupBy a bit more to take a "reduce function". This function would have type Function<List<E>, Integer> . The idea is that function would know how to turn a List into a single number. In the case of CountBy, it would count the elements in the List, and you could simply provide a different Function<List<E>, Integer> to support other cases like MaxBy, MinBy, AverageBy, etc.

16.3 Real query languages

Your operators are related to common operators in query languages for databases. For example, in the widely-used ***SQL*** query language:

- GROUP BY is similar to GroupBy
- COUNT is similar to CountBy and Count
- SELECT is similar to Transform
- SELECT DISTINCT is similar to Distinct
- WHERE is similar to Filter
- JOIN is similar to MapJoin
- UNION ALL is similar to Concatenate

17 Query 10 hints

You might choose to reveal these only one at a time, peeking at the next one if you get stuck.

- Do something that is similar to Algorithm 3 from Query 10 in Homework 6. The rest of the hints, give the structure of the query, one step at a time.

- Filter then GroupBy to collect certain FlightRecords by their destination

- Filter then GroupBy to collect certain FlightRecords by their origin

- MapJoin those two maps

- Transform, whose function has type
  Function<Pair<List<FlightRecord>,List<FlightRecord>>, Iterable<String>>

- Flatten

- Distinct