# CODE REVIEW A5

**Authors:** Aditya, Apoorv, Dhvani

**Reviewers:** Vineet, Sharad, Manthan

**Code Quality and Design:**

1. Classes could have been segregated into packages.
2. A lot of methods are very big chunks of code which has created some amount of confusion and redundancy.
   For example, the 'reduce' method in 'FilterReduce'.
   a. It should have been broken down using helper methods.
   b. The method is also very complex, there is an 'if' condition in an 'if' condition in a 'for' loop in an 'if' condition and this pattern is repeated throughout this method.
   c. In the following code snippet, there are 2 similar if conditions one after the other:

```
if (flag) {
    Double cfDouble = Double.parseDouble(cancelledFraction);
    if (cfDouble > 0.2){
        flag = false;
    }
}
if (flag) {
    if (actualDeptTimeInMins - actualArrTimeInMins > MINLAYOVERINMINS &&
        actualDeptTimeInMins - actualArrTimeInMins < MAXLAYOVERINMINS
        && firstHopCancelled == 0 && secondHopCancelled == 0) {
        context.getCounter(Correctness.CORRECT).increment(1);
    }
}
```

   A better approach would be to do something on the lines of:

```
if (flag) {
        helper1(); // executes first if(flag) condition related code
        helper2(); // executes second if(flag) condition related code
```

}

3. *HopReducer.java*

The Hop Reducer outputs *Text* as value. To construct this output value, *getFlightResult* function is used which constructs a string with CSV separators between fields (starting from line 62). Here's an excerpt:

*result[0] = year1 + CSV_SEP + month1 + CSV_SEP + day1 + CSV_SEP*

*+ deptime1 + CSV_SEP + arrivalTime1 + CSV_SEP + airline1 + CSV_SEP*

*+ origin1 + CSV_SEP + destination1 +CSV_SEP + actualArrivalTime1 + CSV_SEP*

*+ cancelled1+ " " + year2 + CSV_SEP + month2 + CSV_SEP + day2 + CSV_SEP*

*+ deptime2 + CSV_SEP + airline2 + CSV_SEP + origin2 + CSV_SEP + destination2*

*+ CSV_SEP + actulDepatureTime2 + CSV_SEP + cancelled2;*

The code above is not easy on the eyes.

Instead of constructing long strings inside the Reducer, authors could create a custom writable with methods to set values in it. This would make the output more readable.

Moreover, it would also make easier for the authors to add or remove fields without changing the Reducer.

The same method (getFlightResult() line: 40) also returns *null* pointer if it can't find two-hop flights. Later on, the calling method of getFlightResult performs a null check (line: 114). Instead of returning null, the already constructed result (which is an array of String of size 2) variable can be returned.

*reduce()* – The reduce method contains nested for loops and within that contains multiple if-else statements, which makes the method very long and unreadable. Using helper functions to simplify the function would be helpful.

4. *HopMapper.java*

The map() method seems to be constructing a huge string using *StringBuilder*. There are sb.append() calls starting from line: 52 to line: 114. The string construction could be separated and put in a different helper method. Moreover, the first value before the first comma separator is encoded as either "First Hop" or "Second Hop". It would be better to abstract away all of this in a separate model class or a writable.

In the method, *isRecordFirstHop* and *isRecordSecondHop*:

```
if(inf.destination.equals(record.get(23))){
   return false;
}
```

```
else{
   return true;
}
```

Author can omit the second else statement. Also, they can also get rid of the curly braces around if statement.

5. In *HopDriver.java* instead of setting a configuration for input file, Distributed Cache could have been used to ensure that the file is distributed to all the mappers.
   In the same file, the job variable is named *hop_calculation* which is inconsistent since the entire code base uses camel casing. Moreover, it's also against Java coding conventions.
6. In *DelayCancelReducer.java* uses variables using classes instead of primitive types. For example consider the variables defined inside the reduce method:

```
Double sumDelays = 0.0;
int total = 0;
Integer totalCancelled = 0;
```

Here, *sumDalays* and *totalCancelled* are initialized as instance of Double and Integer respectively, instead of primitive classes double and int. This could result in a lot of auto-boxing when these variables are later accessed and changed (in this case incremented), which could probably lead to performance issues. It would be safe to stick to primitive types for such kind of usages, since the variables are only used for assigning and incrementing values.

**Report:**

1. Report consisted of 3 experimental queries and their results were depicted graphically, although their scores were not displayed. Also there seems to be no code to calculate the scoring part.
2. The report also included a flow diagram of the system which was helpful.

**Run:**

1. Makefile does not have a setup method. Had to perform setup manually.
2. After following the instructions to change the input folder names in the makefile, the job still threw an error: "Exception in thread "main" java.io.FileNotFoundException: File does not exist: /user/vineet/inputFile/inputs"
3. The folder inputFile/inputs had been created in the root directory of the project as required by the makefile however the actual code was expecting some other path.
4. I did not investigate further as I had already performed the setup on my own.
5. Code was not executed.