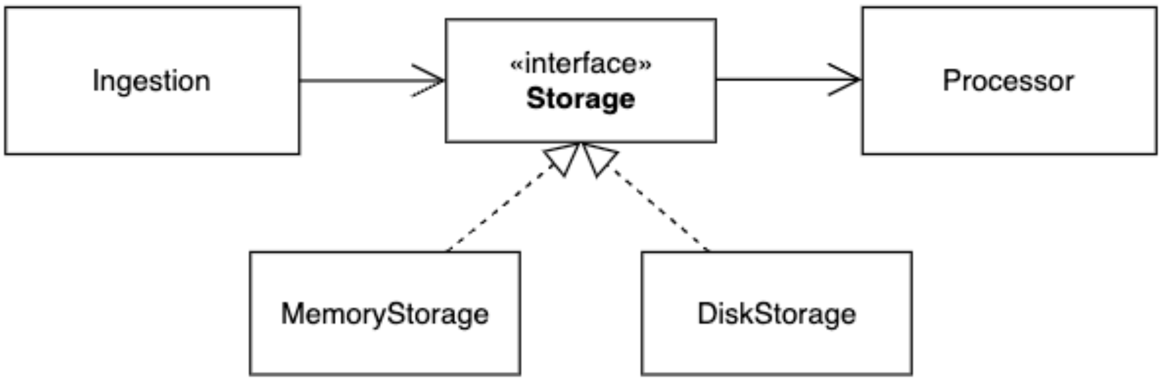
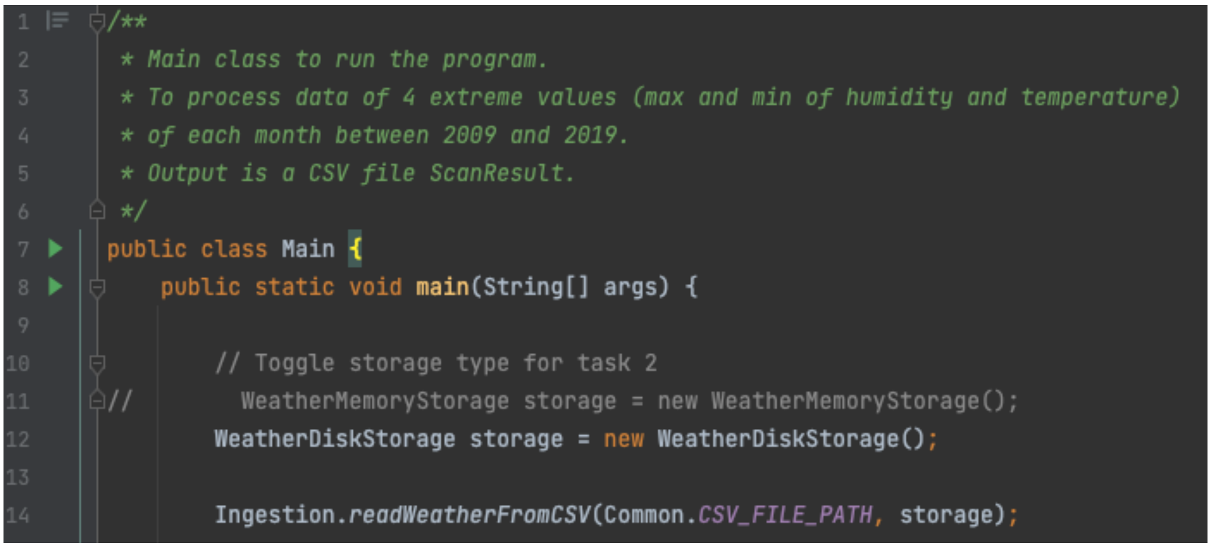
Wincent Tjoi
N2101669E
CZ4123 Project 1 Report



Big overview architecture is shown above (not all classes are shown).



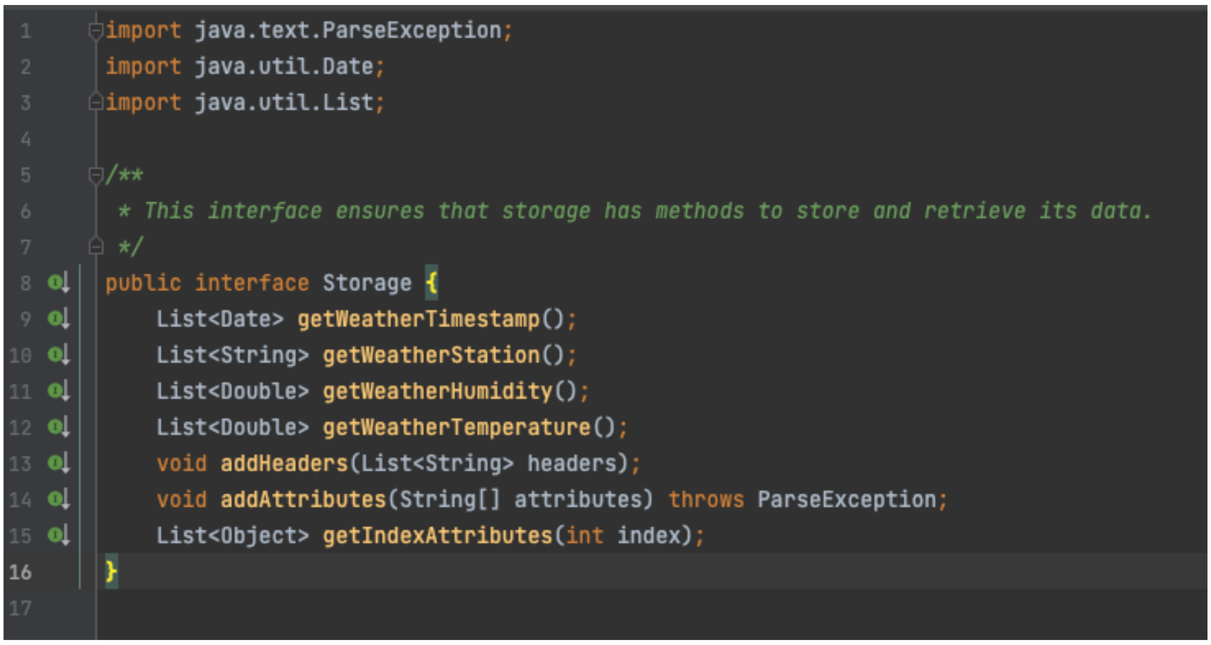
```
1  /**
2   * Main class to run the program.
3   * To process data of 4 extreme values (max and min of humidity and temperature)
4   * of each month between 2009 and 2019.
5   * Output is a CSV file ScanResult.
6   */
7  public class Main {
8      public static void main(String[] args) {
9
10         // Toggle storage type for task 2
11 //        WeatherMemoryStorage storage = new WeatherMemoryStorage();
12         WeatherDiskStorage storage = new WeatherDiskStorage();
13
14         Ingestion.readWeatherFromCSV(Common.CSV_FILE_PATH, storage);
```

Note: To change between storing data in memory or disk (task 1 vs task 2), we have to use line 11 or 12 in Main class to toggle the storage type between WeatherMemoryStorage and WeatherDiskStorage.

Data Storage

Data ingestion is responsible to parse the csv input file and add the data into storage.

There are 2 different types of data storage: MemoryStorage and DiskStorage.
An alternative to this implementation would be Memory Storage to be associated with Disk Storage, such that it can invoke a method to save the data into this. However, in this exercise, having 2 different storage classes to implement a storage interface is used for simplicity in changing between task 1 and task 2.



```java
import java.text.ParseException;
import java.util.Date;
import java.util.List;

/**
 * This interface ensures that storage has methods to store and retrieve its data.
 */
public interface Storage {
    List<Date> getWeatherTimestamp();
    List<String> getWeatherStation();
    List<Double> getWeatherHumidity();
    List<Double> getWeatherTemperature();
    void addHeaders(List<String> headers);
    void addAttributes(String[] attributes) throws ParseException;
    List<Object> getIndexAttributes(int index);
}
```

How to handle exceptions (empty entries, absent month, etc.)?

The code snippet above shows all the APIs that needs to be implemented by Storage.
Storage:AddAttributes(String[]) handles the necessary changes needed to their types, and checks whether the input is not empty. For example, if “M” is found for temperature or humidity column, a double.NaN is used to indicate that it is an empty field.

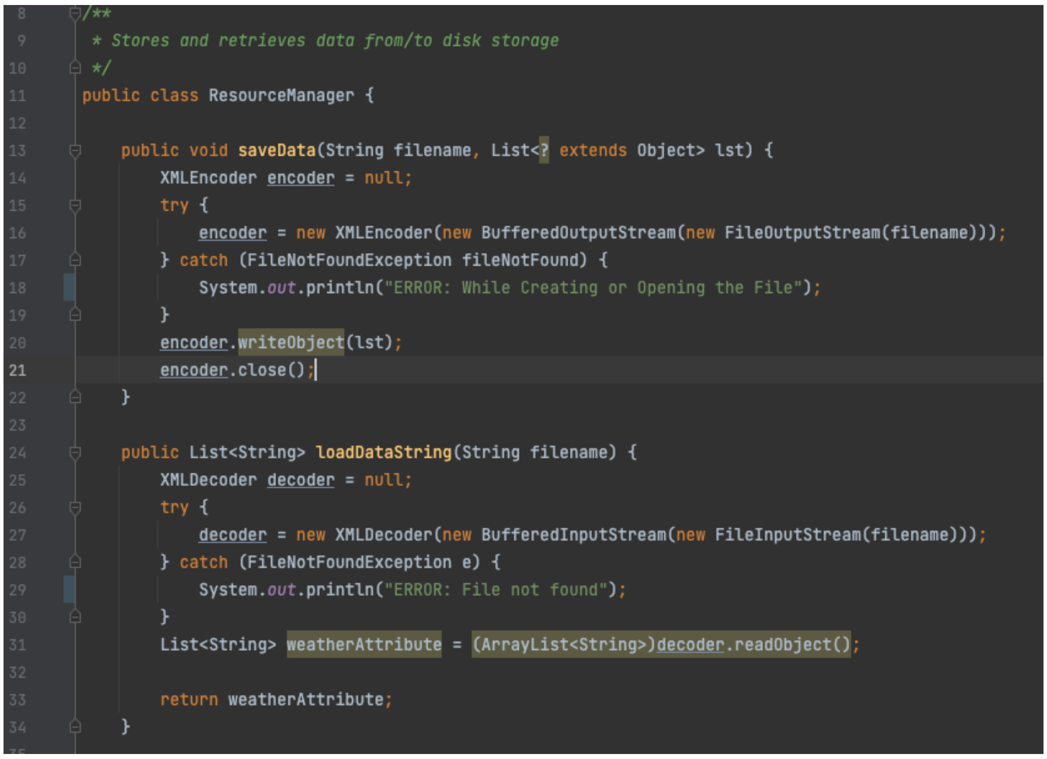
How to store the data in the column-store approach in the main memory (Task 1) and/or disk (Task 2)?

```java
private List<Integer> weatherId = new ArrayList<Integer>();
private List<Date> weatherTimestamp = new ArrayList<Date>();
private List<String> weatherStation = new ArrayList<String>();
private List<Double> weatherTemperature = new ArrayList<Double>();
private List<Double> weatherHumidity = new ArrayList<Double>();
```

Column-store approach is done in the main memory by storing them in ArrayList data structure and each variable represents a column. This is possible as Java ArrayList uses a contiguous memory usage as how it is implemented.

How to read and write the input/output files?

Storing them in disk would have the similar approach, just that the data will be stored persistently in a file (.xml, .txt, .csv, etc) and retrieved from this file again when needed as an additional step. In my implementation, a ResourceManager class is responsible to save and load data persistently into an .xml format. You may refer to the image below.



```java
/**
 * Stores and retrieves data from/to disk storage
 */
public class ResourceManager {

    public void saveData(String filename, List<? extends Object> lst) {
        XMLEncoder encoder = null;
        try {
            encoder = new XMLEncoder(new BufferedOutputStream(new FileOutputStream(filename)));
        } catch (FileNotFoundException fileNotFound) {
            System.out.println("ERROR: While Creating or Opening the File");
        }
        encoder.writeObject(lst);
        encoder.close();
    }

    public List<String> loadDataString(String filename) {
        XMLDecoder decoder = null;
        try {
            decoder = new XMLDecoder(new BufferedInputStream(new FileInputStream(filename)));
        } catch (FileNotFoundException e) {
            System.out.println("ERROR: File not found");
        }
        List<String> weatherAttribute = (ArrayList<String>)decoder.readObject();

        return weatherAttribute;
    }
```

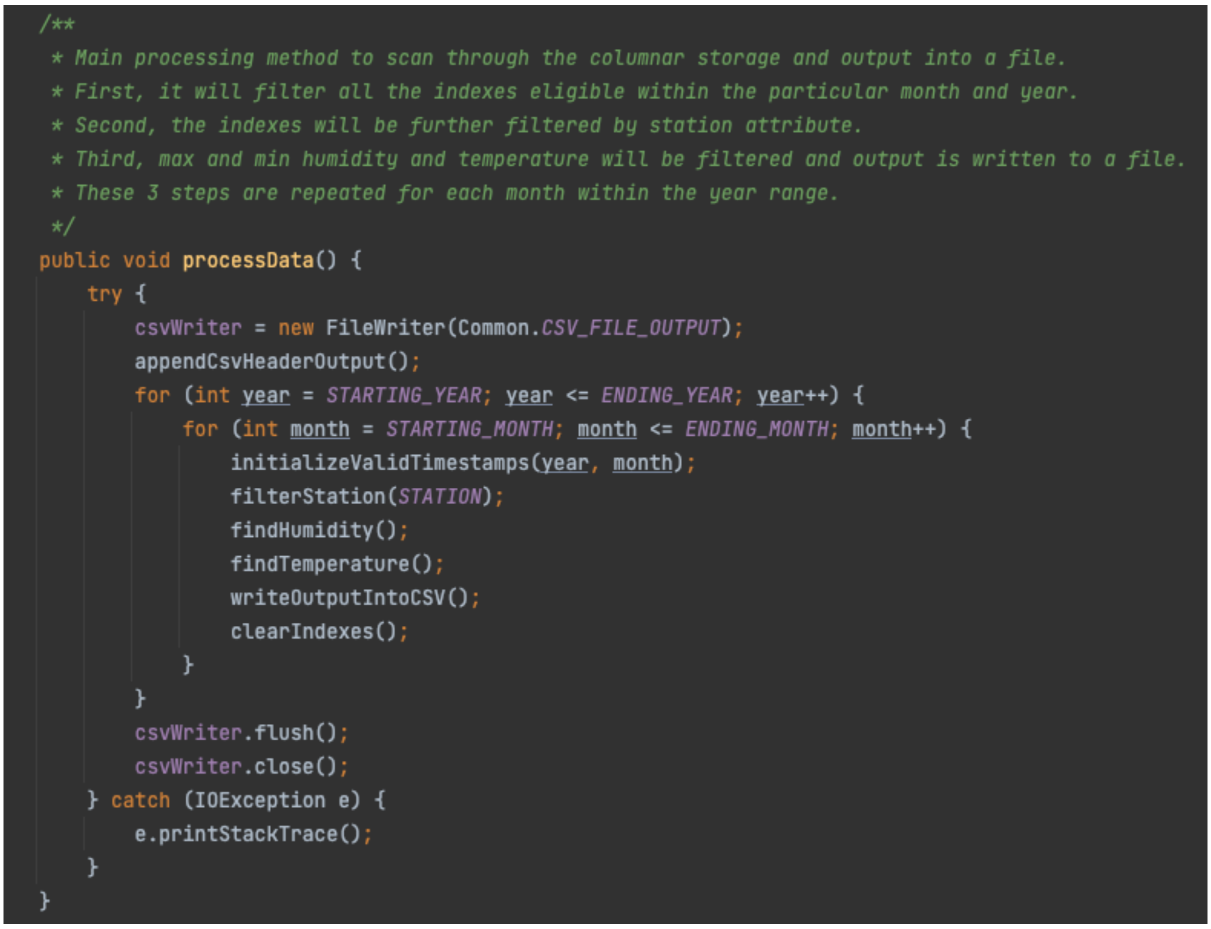
How to design data columns for efficient processing?

Indexes (Hash/B+ Tree) can be used to further optimize processing data.

Note: Indexes are not implemented in this exercise.

Data Processing

How to scan columns according to task conditions?



```java
/**
 * Main processing method to scan through the columnar storage and output into a file.
 * First, it will filter all the indexes eligible within the particular month and year.
 * Second, the indexes will be further filtered by station attribute.
 * Third, max and min humidity and temperature will be filtered and output is written to a file.
 * These 3 steps are repeated for each month within the year range.
 */
public void processData() {
    try {
        csvWriter = new FileWriter(Common.CSV_FILE_OUTPUT);
        appendCsvHeaderOutput();
        for (int year = STARTING_YEAR; year <= ENDING_YEAR; year++) {
            for (int month = STARTING_MONTH; month <= ENDING_MONTH; month++) {
                initializeValidTimestamps(year, month);
                filterStation(STATION);
                findHumidity();
                findTemperature();
                writeOutputIntoCSV();
                clearIndexes();
            }
        }
        csvWriter.flush();
        csvWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Processor:processData() is the main method to scan the column data and handle parsing output into a file. It scans through month and year that are within the acceptable range and further filter stations. Lastly, humidity and temperature will be processed and output to be written to a file. This process then repeats for the next month and/or year.

How to decide and record maximum and minimum values?

1. Scan through the filtered indexes and store the maximum/ minimum value of temperature/ humidity.
2. Scan the indexes again that contain the maximum/ minimum value, and store the index in the correct memory/disk slot. (Time complexity: O(2N), N being the total rows of a column).

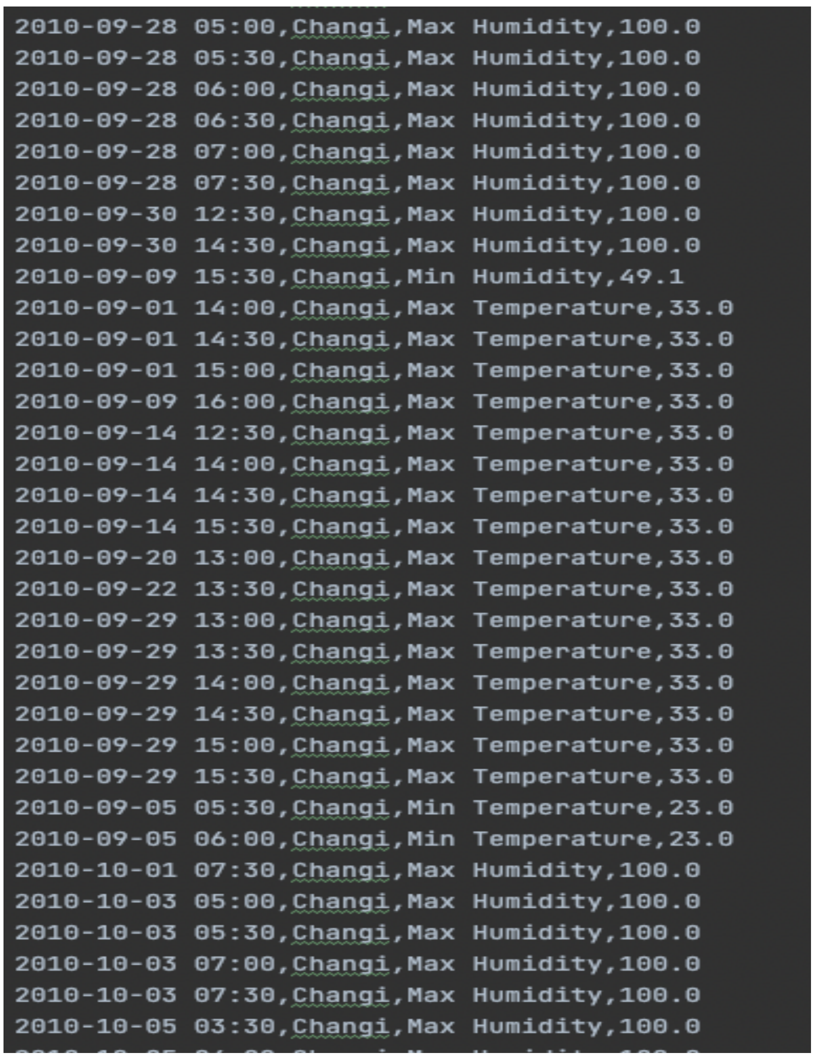
How to improve the efficiency in scanning columns?

Knowing that the query of the month and year would always be in the range of 2009 and 2019, we can store the indexes of the valid months between 2009 and 2019 in a temporary array. Other filters will then proceed from this filtered range of valid months instead of the entire SingaporeWeather input again.
However, this processing method is not implemented as my code implementation tries to follow closely to the SQL query provided in the project task, where Tab1 stores a month after quering from the entire file, and then repeats for the other months.

Experiment Result

A screenshot that your program executes and outputs results successfully

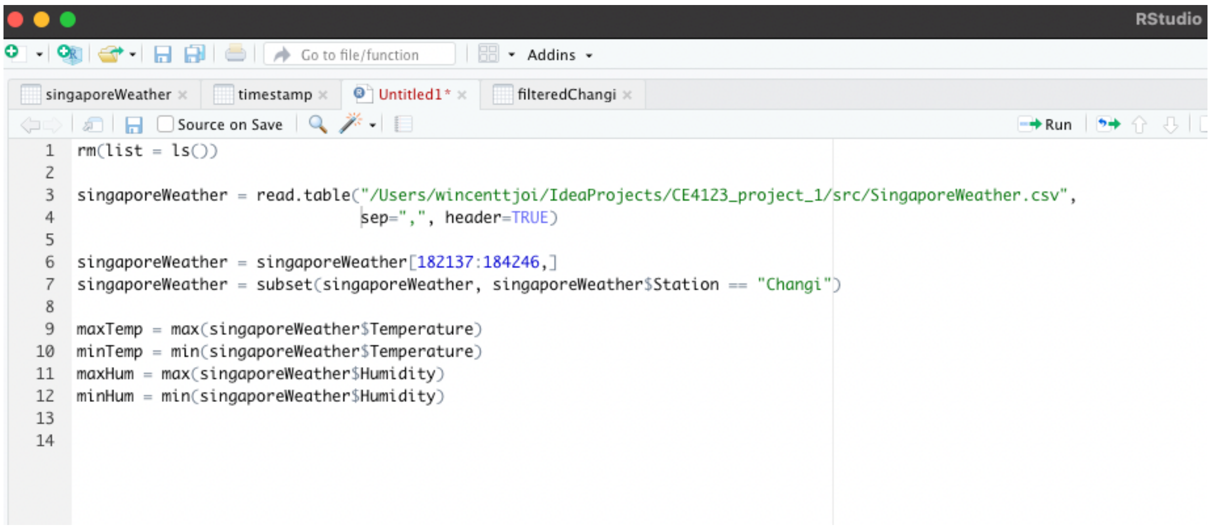
Referring to the image below, we can see that in 2009-09 there is a set of date(s) with Max/Min Humidity/Temperature respectively.
Note: This image is non-exhaustive/non-representative of the entire file output.



```
2010-09-28 05:00,Changi,Max Humidity,100.0
2010-09-28 05:30,Changi,Max Humidity,100.0
2010-09-28 06:00,Changi,Max Humidity,100.0
2010-09-28 06:30,Changi,Max Humidity,100.0
2010-09-28 07:00,Changi,Max Humidity,100.0
2010-09-28 07:30,Changi,Max Humidity,100.0
2010-09-30 12:30,Changi,Max Humidity,100.0
2010-09-30 14:30,Changi,Max Humidity,100.0
2010-09-09 15:30,Changi,Min Humidity,49.1
2010-09-01 14:00,Changi,Max Temperature,33.0
2010-09-01 14:30,Changi,Max Temperature,33.0
2010-09-01 15:00,Changi,Max Temperature,33.0
2010-09-09 16:00,Changi,Max Temperature,33.0
2010-09-14 12:30,Changi,Max Temperature,33.0
2010-09-14 14:00,Changi,Max Temperature,33.0
2010-09-14 14:30,Changi,Max Temperature,33.0
2010-09-14 15:30,Changi,Max Temperature,33.0
2010-09-20 13:00,Changi,Max Temperature,33.0
2010-09-22 13:30,Changi,Max Temperature,33.0
2010-09-29 13:00,Changi,Max Temperature,33.0
2010-09-29 13:30,Changi,Max Temperature,33.0
2010-09-29 14:00,Changi,Max Temperature,33.0
2010-09-29 14:30,Changi,Max Temperature,33.0
2010-09-29 15:00,Changi,Max Temperature,33.0
2010-09-29 15:30,Changi,Max Temperature,33.0
2010-09-05 05:30,Changi,Min Temperature,23.0
2010-09-05 06:00,Changi,Min Temperature,23.0
2010-10-01 07:30,Changi,Max Humidity,100.0
2010-10-03 05:00,Changi,Max Humidity,100.0
2010-10-03 05:30,Changi,Max Humidity,100.0
2010-10-03 07:00,Changi,Max Humidity,100.0
2010-10-03 07:30,Changi,Max Humidity,100.0
2010-10-05 03:30,Changi,Max Humidity,100.0
```

Evaluations that the output results are correct

R program is used to identify the minimum and maximum temperature and humidity of a particular month respectively.
The result is then cross-referred with ScanResult to see if they have the same output.



```
1   rm(list = ls())
2
3   singaporeWeather = read.table("/Users/wincenttjoi/IdeaProjects/CE4123_project_1/src/SingaporeWeather.csv",
4                                 sep=",", header=TRUE)
5
6   singaporeWeather = singaporeWeather[182137:184246,]
7   singaporeWeather = subset(singaporeWeather, singaporeWeather$Station == "Changi")
8
9   maxTemp = max(singaporeWeather$Temperature)
10  minTemp = min(singaporeWeather$Temperature)
11  maxHum = max(singaporeWeather$Humidity)
12  minHum = min(singaporeWeather$Humidity)
13
14
```

Note: Sampling is used for R testing, one month at a time. Referring to the image above (line 6), hard-coded numbers (of a particular month range) are used instead of proper implementation of year range comparison. This is to save time on testing as the data given is too large, causing conversion and comparison of dates to be unbelievably slow. Since this module does not focus on R, manual check comparison between a certain month-year output is done between R output with the Java implementation program.