Team Project

Class design clarification Manual

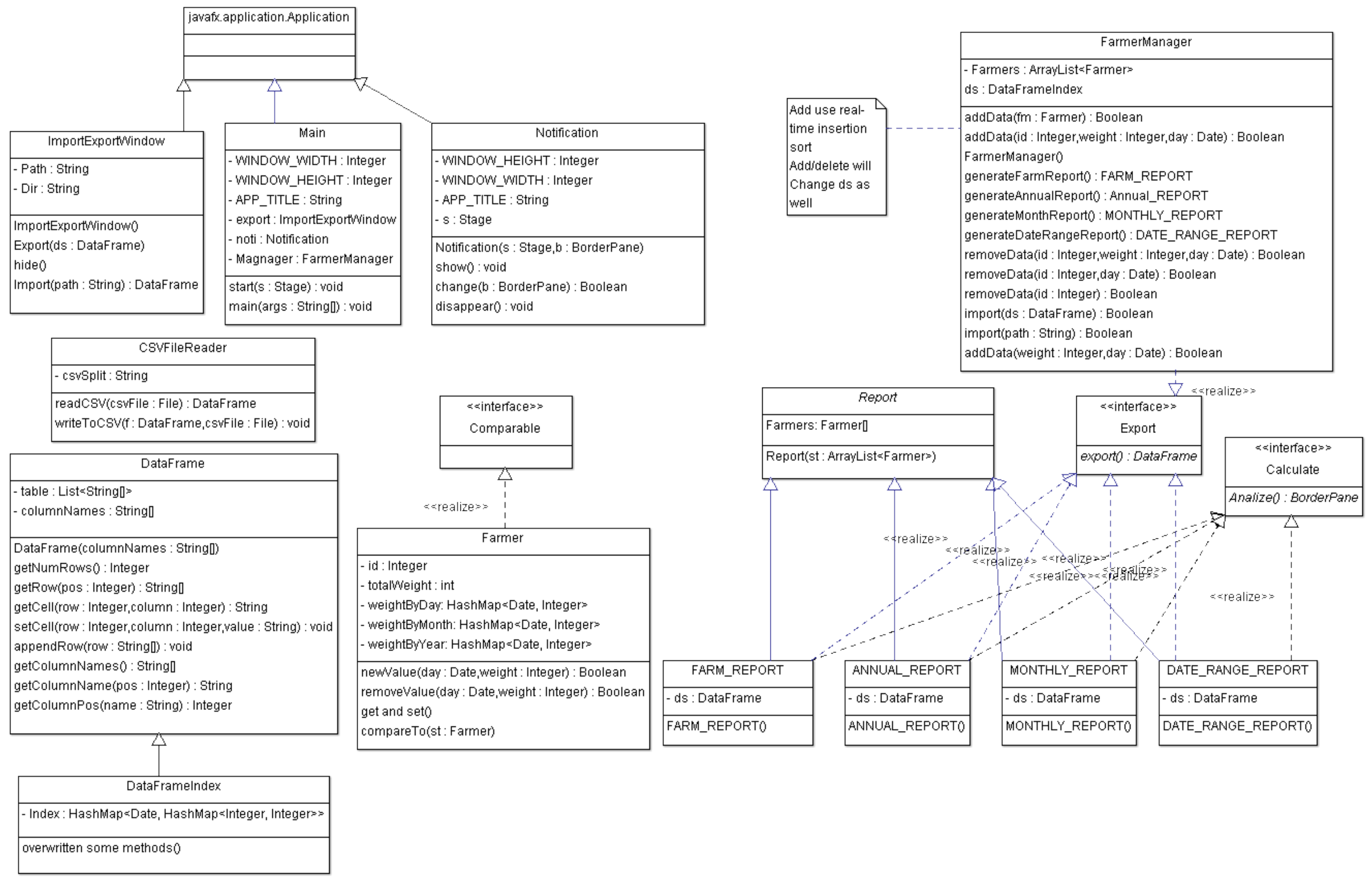
Shaokang Jiang
sjiang97@wisc.edu

May 12, 2020

Contents

1 General idea

The structure and relationship of each class has shown in the UML diagram at Figure below.


javafx.application.Application
ImportExportWindow
- Path : String
- Dir : String
ImportExportWindow()
Export(ds : DataFrame)
hide()
Import(path : String) : DataFrame
Main
- WINDOW_WIDTH : Integer
- WINDOW_HEIGHT : Integer
- APP_TITLE : String
- export : ImportExportWindow
- noti : Notification
- Magnager : FarmerManager
start(s : Stage) : void
main(args : String[]) : void
Notification
- WINDOW_HEIGHT : Integer
- WINDOW_WIDTH : Integer
- APP_TITLE : String
- s : Stage
Notification(s : Stage,b : BorderPane)
show() : void
change(b : BorderPane) : Boolean
disappear() : void
Add use real-time insertion sort Add/delete will Change ds as well
FarmerManager
- Farmers : ArrayList<Farmer>
ds : DataFrameIndex
addData(fm : Farmer) : Boolean
addData(id : Integer,weight : Integer,day : Date) : Boolean
FarmerManager()
generateFarmReport() : FARM_REPORT
generateAnnualReport() : Annual_REPORT
generateMonthReport() : MONTHLY_REPORT
generateDateRangeReport() : DATE_RANGE_REPORT
removeData(id : Integer,weight : Integer,day : Date) : Boolean
removeData(id : Integer,day : Date) : Boolean
removeData(id : Integer) : Boolean
import(ds : DataFrame) : Boolean
import(path : String) : Boolean
addData(weight : Integer,day : Date) : Boolean
CSVFileReader
- csvSplit : String
readCSV(csvFile : File) : DataFrame
writeToCSV(f : DataFrame,csvFile : File) : void
<<interface>>
Comparable
<<realize>>
Report
Farmers: Farmer[]
Report(st : ArrayList<Farmer>)
<<interface>>
Export
export() : DataFrame
<<interface>>
Calculate
Analize() : BorderPane
DataFrame
- table : List<String[]>
- columnNames : String[]
DataFrame(columnNames : String[])
getNumRows() : Integer
getRow(pos : Integer) : String[]
getCell(row : Integer,column : Integer) : String
setCell(row : Integer,column : Integer,value : String) : void
appendRow(row : String[]) : void
getColumnNames() : String[]
getColumnName(pos : Integer) : String
getColumnPos(name : String) : Integer
Farmer
- id : Integer
- totalWeight : int
- weightByDay: HashMap<Date, Integer>
- weightByMonth: HashMap<Date, Integer>
- weightByYear: HashMap<Date, Integer>
newValue(day : Date,weight : Integer) : Boolean
removeValue(day : Date,weight : Integer) : Boolean
get and set()
compareTo(st : Farmer)
FARM_REPORT
- ds : DataFrame
FARM_REPORT()
ANNUAL_REPORT
- ds : DataFrame
ANNUAL_REPORT()
MONTHLY_REPORT
- ds : DataFrame
MONTHLY_REPORT()
DATE_RANGE_REPORT
- ds : DataFrame
DATE_RANGE_REPORT()
DataFrameIndex
- Index : HashMap<Date, HashMap<Integer, Integer>>
overwritten some methods()

Some basic agreement:

- Though the farm_id field in sample is in String format as farm 0. Since it is in the format of ”farm xx” in any example, so we remove the farm part and remain the xx part. Reason to do this is for optimizing speed when sort and easier to index.
- Definition for different weight in farmer class. WeightByday, weightbymonth, etc. Shown in 4.1
- **Invalid data** Means data is not in the expected format or class is not the same. In most case, it means using Class.forName(c).isInstance(obj) to test
- The main GUI window should represent the overall data
- Most of classes should be extendable, which means it should also works for multiple other classes and other people are able to use them as well.
- Each Farmer is using its id as the unique identification.

In the following document, different classes will be discussed. The description and function of each class is also discussed below. The detailed implementation is up to each person. I organize them as groups of classes I planed before. Other basic class is not discussed below as they are only few of lines and they are provided at 4.1

Some sections are small because most people should familiar with them as they are content in or before CS300. And all files should be in application package. The workload of each section might be not even. So the words like one person is responsible might not be strictly correct. And we could work together for each section.

2 Basic component

In the following section, some basic component will be discussed. One person is responsible for this section. Those are basic components this program will be interprete on. The detailed function of each class will be discussed in each class.

Classes: DataFrame, Farmer, CSVFileReader, DataFrameIndex

2.1 DataFrame

A class for storing a table format as a CSV file. You could think each line in CSV as a row and each column in CSV as a cell. This is the basic component that will be used to handle the import and out put function. When data is imported, it will first be translated to DataFrame and then be added into the manager.

In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import java.util.List;

/**
 * A class for storing a table format as a CSV file. You could think each line in CSV as a row
 * and each column in CSV as a cell
 *
 * @author
 *
 */
public class DataFrame {

  //some sample private data field. You may change them.
  //All of them should be in private access
  private String[] column;
  private List<Object[]> rows;
  private Object[] dataType;
  /**
   * Create a new DataFrame with a given list of columns. column could be thought as the header
   * of your table. For example,
   * if csv file is:
   * Time,rType,id,Speed,lane,u
   * 0,MainRoad,200,58,0,660.09908
   * 0,MainRoad,201,58,1,625.52486
   * 0,MainRoad,202,58,2,590.95306
   * 0,MainRoad,203,58,0,556.38140
   *
   * Then,
   * The pass in column could be [Time,rType,id,Speed,lane,u]
   *
   * For the dataType field, It will be used to validate if the future passed in value is in the
   * correct data type or not. The passed in dataType only needs to be in the same class as data
   * needs. It doesn;t need to be exactly the same as the first row. But usually the first row is
   * being passed in for representing this feature.
   *
   * For the example above, the passed in datatype could be [0,MainRoad,200,58,0,660.09908]
   *
   * You will need to use this to determine if the future passed in row is in good format or not
   * in the append row, setcell method.
   *
   * And before setup, you need to transform those value from String to their specific format. In
   * this project case, first one should be transformed to java.util.Date second one should be
   * transformed to int, third one should be transformed to int
   *
   * @param column the names of the columns for the new DataFrame
   */
  public DataFrame(String[] column, Object[] dataType) {

  }

  /**
   * Return the nth row.
   *
   * @param pos the nth row that we want to return, first row is row 0
   * @return the row String[]
   */
  public Object[] getRow(int pos) {

  }

  /**
   * Get the value in the specific cell. To define a cell, it is using the col and row position.
   *
   * @param row position of the row first row is at pos 0
   * @param column column's position, start from 0
   * @return the value at the specific cell
   */
  public Object getCell(int row, int column) {

  }

  /**
   * Change data of a specific cell to the value passed in
   *
   * At the same time, you need to test if the passed in value is in correct dataType or not.
   *
   * For datatype, refer to the dataType field defined when initialized
   *
   * For testing if value is in correct dataType, you could use a reference at geeksforgeeks
   * I forget the specific page
   * // This method tells us whether the object is an
   * // instance of class whose name is passed as a
   * // string 'c'.
   * public static boolean fun(Object obj, String c) throws ClassNotFoundException
   * { return Class.forName(c).isInstance(obj); }
   *
   * You need to handle the exception correcctly except IllegalArgumentException.
   *
   * @param row position of the row first row is at pos 0
```

```
     * @param column column's position, start from 0
     * @param value the new value for this cell
     * @exception IllegalArgumentException if passed in value doesn;t meet the requirement of
     * data at that position
     */
    public void setCell(int row, int column, Object value) throws IllegalArgumentException {

    }

    /**
     * Appends a new row to the DataFrame.
     *
     * At the same time, you need to test if the passed in value is in correct dataType or not.
     *
     * For datatype, refer to the dataType field defined when initialized
     *
     * For testing if value is in correct dataType, you could use a reference at geeksforgeeks
     * I forget the specific page
     * // This method tells us whether the object is an
     * // instance of class whose name is passed as a
     * // string 'c'.
     * public static boolean fun(Object obj, String c) throws ClassNotFoundException
     * { return Class.forName(c).isInstance(obj); }
     *
     * You need to handle the exception correcctly except IllegalArgumentException.
     *
     * @param row the new row given as a String[]
     * @throws IllegalArgumentException if the row does not have the same number of columns as the
     * DataFrame should have or if passed in value doesn;t meet the requirement of
     * data at that position
     */
    public void appendRow(Object[] row) throws IllegalArgumentException {

    }


    /**
     * Returns the column names of this DataFrame.
     *
     * @return the column names
     */
    public String[] getColumnNames() {

    }

    /**
     * Return the name of the column at the passed in position.
     *
     * @param pos the position of the column
     * @return the name of the column at that position
     */
    public String getColumnName(int pos){
    }

    /**
     * Returns the position of the column with the given name.
     *
     * @param name the name of the column
     * @return the column's position
     */
    public int getColumnPos(String name) {

    }

  }
```

2.2 Farmer

Well, this one is easy to see to be a farmer. It stores information of a farmer. For the private field in this program: Those fields are designed for making the analize part get data quicker

- **weightByDay** This contains milk this farmer sent out every day, access the hashmap by using Date(”2019-1-1”), Date(”2019-1-2”).
- **weightByMonth** This contains milk this farmer sent out every month, access the hashmap by using Date(”2019-1-1”), Date(”2019-2-1”). The first day of each month.
- **weightByYear** This contains milk this farmer sent out every month, access the hashmap by using Date(”2019-1-1”), Date(”2020-1-1”). The first day of each year.

In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import java.util.Date;
import java.util.HashMap;

public class Farmer implements Comparable<Farmer>{

  private int id;
  private int totalWeight;
  private HashMap<Date, Integer> weightByDay;
  private HashMap<Date, Integer> weightByMonth;
  private HashMap<Date, Integer> weightByYear;

  /**
   * Should add this new value to each field. If the entry in the
   * hashmap already exist, then increment it by the {@value weight} amount.
   *
   * For e.g.
   *
   * If there is a line of data:
   * day = 2019-1-2, weight = 10.
   *
   * The you need to do,
 * weightByDay.get(Date(2019-1-2)) != null
 * do: weightByDay.get(Date(2019-1-2)) += weight
 * else create this entry and set the value to weight.
 * Then find the weightByMonth:
 * weightByMonth.get(Date(2019-1-1)) != null
 * do things like before
 *
 * same thing should be done for year
 *
 * and totalweight should also be incremented.
 *
 * Those are just pseudocode, not javacode
 *
 * @param day
 * @param weight
 * @return
 */
public Boolean newValue(Date day, int weight) {

}

/**
 * Should remove this new value to each field. If the entry in the
 * hashmap already exist, then decrease it by the {@value weight} amount.
 *
 * the finding procedure is the same as before
 * One thing dfferent is that when not found method should return false
 *
 * and totalweight should also be decrement.
 *
 *
 * @param day
 * @param weight
 * @return
 */
public Boolean removeValue(Date day, int weight) {

}

/**
 *
 * @return the id of this farmer
 */
public int getId() {

}

/**
 *
 * @return the totalweight sent by this farmer
 */
public int getTotalWeight() {
    }

    /**
     *
     * @return weightByDay
     */
    public HashMap<Date, Integer> getWeightByDay(){

    }

    /**
     *
     * @return weightByMonth
     */
    public HashMap<Date, Integer> getWeightByMonth(){

    }

    /**
     *
     * @return weightByYear
     */
    public HashMap<Date, Integer> getWeightByYear(){

    }

    @Override
    /**
     * Compare with another object based on their id.
     *
     * @return this.getId() - o.getId()
     */
    public int compareTo(Farmer o) {

    }

}
```

2.3 CSVFileReader

This will handle all File I/O required by the assignment. It will read from CSV file to dataframe or dataframe to CSV file using different methods.

In more detail, format looks like below. Feel free add any private field or method.

```
package application;


/**
 * Handle read/write with CSV file
 *
 */
public class CSVFileReader {
    /**
     * Create a DataFrame based on the passed in CSV file.
     *
     * You can treat the first row after header in csv file as the default dataType
     *
     * You need to prompt a window if dataType if is incorrect or fail to be converted
     *
     * For example,
     * correct format:
     * date,farm_id,weight
     * 2019-1-1,Farm 0,6760
     * Wromg data:
     * 2019-1-1,ZU4XV2,11924
     *
     * You won't be able to convert or interpreted it to the format of xx(in Integer)
     * At this time, you should prompt a dialog to tell user data file has error
     * and return null.
     *
     * @param csvFile the input CSV file
     * @return the DataFrame generated
     * @throws Exception if CSV file contains error:
     * e.g. DataType in this row is not correct as it should be.
     *
     *
     */
    public DataFrame readCSV(File csvFile) throws Exception {

    }

  /**
   * Write a DataFrame to a CSV file.
   *
   * Remember the agreement that: Though the farm_id field in sample is in String format as farm
   *  0. Since it is in the format of "farm xx" in any example, so we remove the farm part and
   * remain the xx part. Reason to do this is for optimizing speed when sort and easier to index.
   *
   * So, for the farmer_id col, you need to turn the number "xx" into "farm xx" and then write to
   * CSV file
   *
   * @param f the DataFrame
   * @param csvFile the file to write
   * @throws Exception if file I/O errors happens
   */
  public void writeToCSV(DataFrame f, File csvFile) throws Exception {

  }

}
```

2.4 DataFrameIndex

This is a special dataframe with the indexing feature to find the position of a farm and a day. It uses the following procedure to find, First layer of hashmap will find the position for farmers data in a specific day by using Date field. Then by using the farmer id, we will be able to find the position of this farmer at a specific day in the list, Then we could use this number to access the list and change the required value.

And remember, this one is extended from DataFrame, so it should meet all requirement dataFrame has. So the javadoc in this file only mention the difference

In more detail, format looks like below. Feel free add any private field or method.

```java
  package application;

import java.util.Date;
import java.util.HashMap;
import java.util.List;

/**
 * A class for storing a table format as a CSV file with quick indexing feature.
 * Still useful as:
 * 1. remove duplicates in imported csv file
 * 2. Easy for FarmerManager to export data.
 * 3. save runtime for manager class. Because user will only add/remove some data.
 * This is not the same as filter. User are not able to filter data in the main GUI
 * as the main GUI is supposed to represent the overall data
 *
 *
 * @author
 *
 */
public class DataFrameIndex extends DataFrame {

  //I think we need to use those variables
  //feel free to add more private data field

  private HashMap<Date, HashMap<Integer, Integer>> Index;

  /**
   * Do whatever the DateFarme is done
   * You need to initialize the Hashmap as well.
   *
   * @param column the names of the columns for the new DataFrame
   */
  public DataFrameIndex(String[] column, Object[] dataType) {

  }

  /**
   *
   * Those step should be done after the step of checking correctness is done.
   * If the information of the given row is already existed in the form,
   * e.g. if you try to access the date and farmid in the hashmap and the return value is not null.
   * Then it means it is existed in the list. By using the int you get, you can access the
   * list and add the gallons of milk to the form at the specific position.
   *
   * If it is not existed in the hashmap, you should add it to the hashmap and store the position
   * in the list to the hashmap.
   *
   * Same as before, If invalid data, need give user a prompt
   *
   * @param row the new row given as a String[]
   * @throws IllegalArgumentException if the row does not have the same number of columns as the
   * DataFrame should have or if passed in value doesn;t meet the requirement of
   * data at that position
   */
  public void appendRow(Object[] row) throws IllegalArgumentException {

  }

  /**
   * Same procedure as before. Difference is that: when I find this element
   * I will set its value directly to this value passed in
   *
   * If invalid data, need give user a prompt
   *
   * @param id the id of the farm
   * @param day the day of the farm
   * @param val
   */
  public void setVal(int id, Date day, int val) {

  }

  /**
   * Same finding procedure as before.
   * This time, you need to remove the entry for this farmer in this day.
   * And if this farmer doesn;t have any day value that still contains
   * gallon data. You do also need to remove the farmer entry.
   *
   * And don't forget to remove the row in rows as well.
   *
   * If invalid data, need give user a prompt
   *
   * @param id
   * @param day
   */
  public void removeRow(int id, Date day){

  }

}
```

3 Manager part

In the following section, The manager part will be discussed. One person is responsible for this section. This is the class to manage all structure.

Classes: FarmerManager

3.1 FarmerManager

Each Farmer is using its id as the unique identification.
In more detail, format looks like below. Feel free add any private field or method.

```java
package application;

import java.util.Date;
import java.util.List;
import javafx.scene.control.TableView;

public class FarmerManager implements Export<Object> {

  //Main class will use them, so it is needed to be in protected
  protected List<Farmer> Farmers;
  protected DataFrameIndex ds;

  protected class Data{
    protected int id;
    protected Date day;
    protected int gallon;
  }
  /**
   * Constructor of this class, will be called once the entire program starting.
   * Need initialize the local private fields to represent them as empty.
   * The columnsNames in ds would be [date,farm_id,weight]
   */
  public FarmerManager() {

  }

  /**
   * Add data to the current Farmers and ds based on imformation in the passed in
   * fm. You need to make use of the methods in Farmers and ds.
   *
   * This would be the easy case, you have a Farmer object now. you need to merge
   * data in this object with any object that has the same index information. If
   * there is no object that has the same indexing information as this one. You
   * should add a new one to the current Farmers and ds.
   *
   * For example, if the pass in fm has fm.id=1 fm.day=2019-01-01 fm.gallon = 111
   * if there is a farm has the same id information. then call the newValue method
   * in farmer with those information.
   *
   * If not, then add this one to the farmers and also append it to the ds. Also
   * remember, you need to put this one in the correct place, which means the
```

```
     * farmers are always in sorted order.
     *
     * I know a Red-black/AVL tree might be better at here, but in order to make this
     * simplier, we just find the correct position and put it at here.
     *
     * @param fm the passed in fm contain all information
     * @return true if added successfully, otherwise(e.g. data is null, etc.) false
     */
    public Boolean addData(Farmer fm) {

    }

    /**
     *
     * Same procedure as before. Same things need to be done as before
     *
     * @param id the id of the farmer
     * @param weight the weight of farmer
     * @param day the day that those gallon of milk is produced.
     * @return true if added successfully, otherwise(e.g. data is null, etc.) false
     */
    public Boolean addData(Integer id,Integer weight,Date day) {

    }

    /**
     * This one just return a Farm_report class, main method will handle the rest stuff.
     *
     * @return a FARM_REPORT class contain current farmers information
     */
    public FARM_REPORT generateFarmReport() {

    }

    /**
     * This one just return a Annual_REPORT class, main method will handle the rest stuff.
     *
     * @return a Annual_REPORT class contain current farmers information
     */
    public Annual_REPORT generateAnnualReport() {

    }

    /**
     * This one just return a DATE_RANGE_REPORT class, main method will handle the rest stuff.
     *
     * @return a DATE_RANGE_REPORT class contain current farmers information
     */
    public DATE_RANGE_REPORT generateDateRangeReport() {

    }

    /**
```

```
     * This one just return a MONTHLY_REPORT class, main method will handle the rest stuff.
     *
     * @return a MONTHLY_REPORT class contain current farmers information
     */
    public MONTHLY_REPORT generateMonthReport() {

    }

    /**
     * remove those weight on a specific day of a farm.
     * Same finding procedure as {@code addData}
     *
     * @param id the id of the farmer
     * @param weight the weight of farmer to be removed
     * @param day the day that those gallon of milk is produced.
     * @return true if added successfully, otherwise(e.g. data is null, etc.) false
     */
    public Boolean removeData(Integer id,Integer weight,Date day) {

    }

    /**
     * Remove the data of a farm at a specific day. You should also remove data from
     * farmers and ds.
     *
     * Hint: Use method {@code removeRow} to remove in ds
     *
     * @param id the id of the farmer
     * @param day day information to be removed
     * @return true if success, false if this data doesn't exist
     */
    public Boolean removeData(Integer id,Date day) {

    }

    /**
     * remove all information of this id.
     * It means the ds will no longer contain information about this id
     * And this id will also be removed from farmers list
     *
     * @param id the id of the farmer
     * @return true if success, false if this data doesn't exist
     */
    public Boolean removeData(Integer id) {

    }

    /**
     *
     * Load data from a ds, loop through each row in ds and add information
     * to the ds in this class and farmers.
     * The way to find has been discussed in addData, and you could make use
     * of addData
```

```java
   *
   *
   * @param ds1
   * @return true if success, false otherwise(ds doesn;t have required
   * format, etc. )
   */
  public Boolean importData(DataFrame ds1) {

  }

  /**
   * Firstly read file from the path to dataframe, then use importData
   * above to load to the program.
   * refer to CSVFileReader and DataFrame for more methods to use
   *
   * If any row in the ds doesn't qualify the requirement, return
   * false and terminate the loading process. This time no data
   * should be loaded to the program.
   *
   * @param path path to the csv file
   * @return true if success, false if not valid data
   */
  public Boolean importData(String path) {

  }

  /**
   *  Add a new farmer with automatic index. the index is determined by
   *  the idx of the last element in {@code farmers}+1.
   *  Then same procedure as previous addData.
   *
   * @param weight the weight of it
   * @param day the day of the data.
   * @return true if success otherwise false(e.g. try catch with error.)
   */
  public Boolean addData(Integer weight, Date day) {

  }

  @Override
  /**
   * This is a special export, basically, directly return ds of this class
   * would be enough
   */
  public DataFrame export(TableView<Object> a) {
    // TODO Auto-generated method stub
    return null;
  }

}
```

4 Analyze part

In the following section, The Analyze part will be discussed. One person is responsible for this section. This is the class to analyze and generate GUI reqport.

Classes: Report, Calculate, Export, FARM_REPORT, ANNUAL_REPORT, MONTHLY_REPORT

4.1 Interface, abstract class

Those classes is provided below. They are good to be used to organize functions.

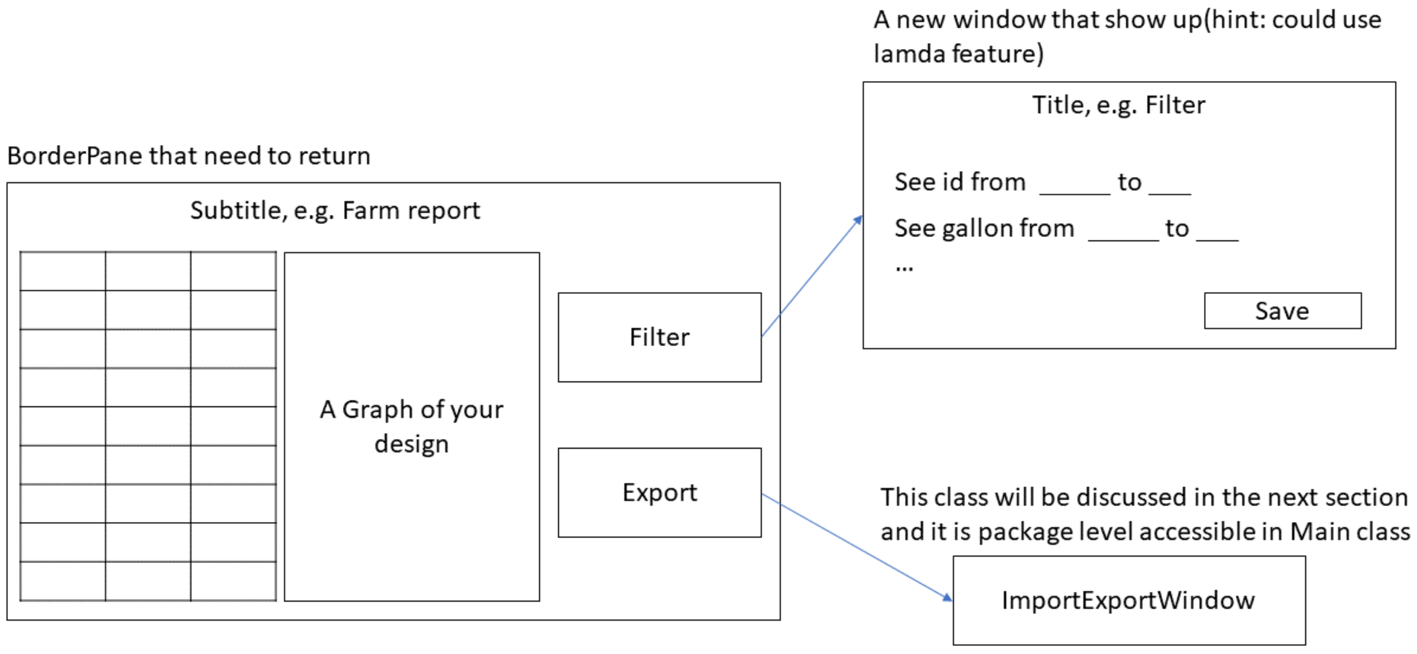
Export interface:

```
package application;

import javafx.scene.control.TableView;

interface Export <S>{

  /**
   * For this method, you need to put the analyze result to a
   * DataFrame, The analyzed result is represented in Tableview a
   * But there are some other special circumstances.
   * For e.g., In FarmerManager, it will be good to return dataFrame
   * directly.
   * @return the dataframe that contains required information in different
   * cases
   */
  DataFrame export(TableView<S> a);

}
```

Calculate interface:

```
package application;

import javafx.scene.layout.BorderPane;

interface Calculate {

  /**
   * In this interface, you need to represent the result of different
   * analyze in the borderpane. Currently we plan to have a TableView,
   * A Graph view, two button to filter or export.
   *
   * When filter is called, Graph and table content should be changed
   * dynamically. It means, once user click save in the filter window
   * the program will change the table and graph in the borderPane window
   *
   * @return the panel contains those information
   */
  BorderPane Analize();

}
```

Each Analize() method should return a borderPane like this: The arrow means when that button is pressed, an interface looks like the connected to component will be shown. Most of places is just a sample. They need to be changed based on the specific class you are working on.


A new window that show up(hint: could use lamda feature)
Title, e.g. Filter
See id from ______ to ___
See gallon from ______ to ___
...
Save
BorderPane that need to return
Subtitle, e.g. Farm report
A Graph of your design
Filter
Export
This class will be discussed in the next section and it is package level accessible in Main class
ImportExportWindow

Abstract Report class:

```
package application;
import java.util.ArrayList;

public abstract class Report {

  private Farmer[] Farmers;

  /**
   * In this method, you need to convert each element of passed in st to
   * an element in Farmers.
   *
   * @param st
   */
  public Report(ArrayList<Farmer> st) {
    Farmers = st.toArray(); //Only pseudocode, need convert
  }


}
```

4.2 FARM_REPORT

This class will be create once user press the farm report button. And the returned panel will be displayed in Notification window once analize is called. (Those are handled by Main class) The export description could be see at 4.1. The description for Analize method could refer to 4.1

In more detail, format looks like below. Feel free add any private field or method.

```java
  package application;

import java.util.ArrayList;
import java.util.Date;
import javafx.scene.control.TableView;
import javafx.scene.layout.BorderPane;

public class FARM_REPORT extends Report implements Calculate, Export<FARM_REPORT.FARM>{

  public FARM_REPORT(ArrayList<Farmer> st) {
    super(st);
    // TODO Auto-generated constructor stub
  }

  @Override
  public DataFrame export(TableView<FARM> a) {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public BorderPane Analize() {
    // TODO Auto-generated method stub
    return null;
  }
  /**
   * This class is used to store information that will be represented in table view in the
   * @author Shaokang Jiang
   *
   */
  protected class FARM{
    protected int id;
    protected Date day;
    protected int Num;
  }

}
```

4.3 ANNUAL_REPORT

This class will be create once user press the ANNUAL report button. And the returned panel will be displayed in Notification window once analize is called. (Those are handled by Main class) The export description could be see at 4.1. The description for Analize method could refer to 4.1

In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import java.util.ArrayList;
import java.util.Date;
import javafx.scene.control.TableView;
import javafx.scene.layout.BorderPane;

public class Annual_REPORT extends Report implements Calculate, Export<Annual_REPORT.Annual>{

  public Annual_REPORT(ArrayList<Farmer> st) {
    super(st);
    // TODO Auto-generated constructor stub
  }

  @Override
  public DataFrame export(TableView<Annual> a) {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public BorderPane Analize() {
    // TODO Auto-generated method stub
    return null;
  }
  /**
   * This class is used to store information that will be represented in table view in the
   * @author Shaokang Jiang
   *
   */
  protected class Annual{
    protected int id;
    protected Date year;
    protected int Num;
  }
}
```

4.4 MONTHLY_REPORT

This class will be create once user press the MONTHLY report button. And the returned panel will be displayed in Notification window once analize is called. (Those are handled by Main class) The export description could be see at 4.1. The description for Analize method could refer to 4.1

In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import java.util.ArrayList;
import java.util.Date;
import javafx.scene.control.TableView;
import javafx.scene.layout.BorderPane;

public class MONTHLY_REPORT extends Report implements Calculate, Export<MONTHLY_REPORT.MONTH>{
  public MONTHLY_REPORT(ArrayList<Farmer> st) {
    super(st);
    // TODO Auto-generated constructor stub
  }

  @Override
  public DataFrame export(TableView<MONTH> a) {
    // TODO Auto-generated method stub
    return null;
  }


  @Override
  public BorderPane Analize() {
    // TODO Auto-generated method stub
    return null;
  }
  /**
   * This class is used to store information that will be represented in table view in the
   * @author Shaokang Jiang
   *
   */
  protected class MONTH{
    protected int id;
    protected Date day;
    protected int Num;
  }
}
```

5 Analyze 2 & Helper GUI part

In the following section, The Analyze 2 & Helper GUI part will be discussed. One person is responsible for this section. This is the class to analyze and generate GUI report.

Classes: DATE_RANGE_REPORT, ImportExportWindow, Notification

5.1 DATE_RANGE_REPORT

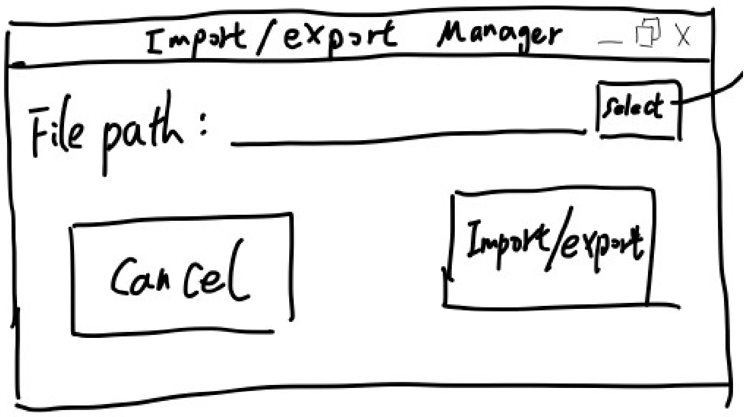
This class will be create once user press the DATE RANGE report button. And the returned panel will be displayed in Notification window once analize is called. (Those are handled by Main class) The export description could be see at 4.1. The description for Analize method could refer to 4.1

In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import java.util.ArrayList;
import java.util.Date;
import javafx.scene.control.TableView;
import javafx.scene.layout.BorderPane;

public class DATE_RANGE_REPORT extends Report implements Calculate,
    Export<DATE_RANGE_REPORT.DATE_RANGE>{

  protected Date start;
  protected Date end;

  public DATE_RANGE_REPORT(ArrayList<Farmer> st, Date start, Date end) {
    super(st);
    // TODO Auto-generated constructor stub
  }

  @Override
  public DataFrame export(TableView<DATE_RANGE> a) {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  /**
   * A little bit different at here. You need to generate the table and the graph based on
   * data range. And when you are registering a filter, you need to make sure the date range
   * are not able to exceed the start and end date range.
   */
  public BorderPane Analize() {
    // TODO Auto-generated method stub
    return null;
  }
  /**
   * This class is used to store information that will be represented in table view in the
   * @author Shaokang Jiang
   *
   */
  protected class DATE_RANGE{
    protected int id;
    protected Date day;
    protected int Num;
  }

}
```

5.2 ImportExportWindow

This class is able to "memorize" last user input by setting the private path variable to an address and this window will only be show or hide. The basic interface looks like this:



In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;

public class ImportExportWindow extends Application {

  protected String path;
  protected Stage s; //need to visit in the future. So start will
  //also need to setup this field.

  @Override
  public void start(Stage s) throws Exception {
    // TODO Auto-generated method stub

  }

  /**
   * put the passed in ds to the path, if path doesn;t contain
   * filename, use export.csv if file already exist, use path+Date.currentTime.
   * @param ds the dataframe being exported
   */
  public void export(DataFrame ds, String path) {
  }

  /**
   * Hide this window
   */
  public void hide() {

  }

  /**
   * Show this window
   * 0 -- import model
   * 1 -- export model
   *
   *
   * @param i int to indicate this is in import model or export model
   */
  public void show(int i) {

  }

  /**
   * Should detect of the path is multiple paths("C:/a.a;C:/b.a")
   * or not If it is, read each csv in each position
   * and merge them into one dataframe then set protected path to the first path in the list
   * , if any of file doesn;t meet the requirement(error file) prompt the user then
   * return null
   *
   *
   * @param path path or path list to files
   * @return the DataFrame contains all information
   */
  public DataFrame Import(String path) {

  }

}
```

5.3 Notification

Notification panel is a framework. It will display report. It might be used for different purpose, for example, representing the repoprt for user to choose. The detailed content depends on borderpane displayed on it.

In the GUI, it looks like at 1:

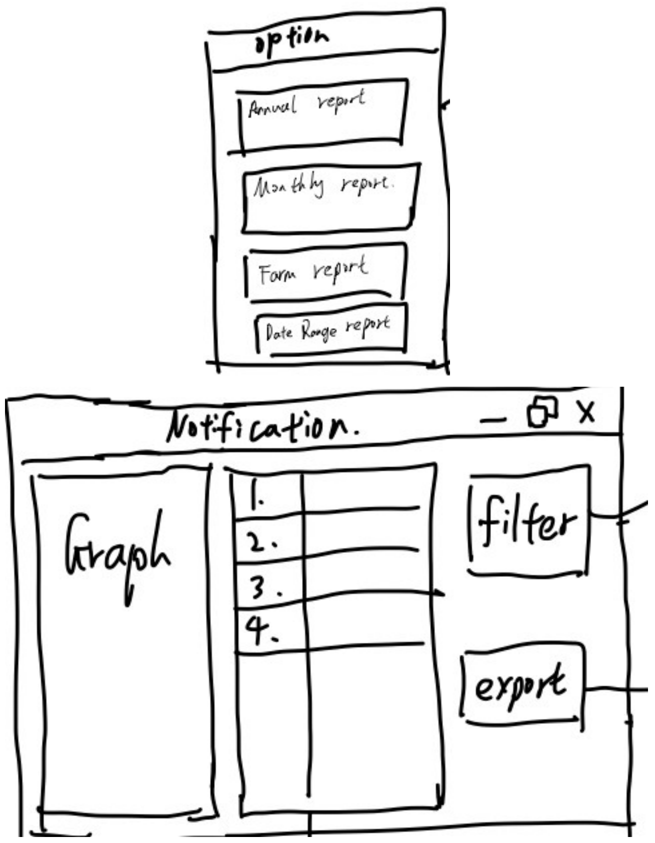


Figure 1: Either one could be in Notification

In more detail, format looks like below. Feel free add any private field or method.

```java
package application;

import javafx.application.Application;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Notification extends Application {

  protected Stage s;

  @Override
  public void start(Stage arg0) throws Exception {
    // TODO Auto-generated method stub

  }

  /**
   * show this window
   */
  public void show() {

  }

  /**
   * hide this window
   */
  public void hide() {

  }

  /**
   * Change the borderpane of the current stage to the input one.
   * Might dynamically change the title, might not as well
   * @param bp
   * @return
   */
  public boolean change(BorderPane bp) {

  }

}
```

6 Main GUI

In the following section, The main GUI will be discussed. One person is responsible for this section. This is the class to Manage all interfaces in the GUI and interpret with user.

Classes: Main

6.1 Main

Lots of button methods are registered and initialized at here in the start function. In the start place, importexportwindow and Notification window will aso be initialized but they will not be shown. FarmerManager also needs to be initialized.

In more detail, format looks like below. Feel free add any private field or method.

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;

public class Main extends Application {

  protected ImportExportWindow importExportWindow;
  protected Notification noti;
  protected FarmerManager Magnager;

  @Override
  public void start(Stage arg0) throws Exception {
    // TODO Auto-generated method stub

  }

  public static void main(String[] args) {

  }
}
```

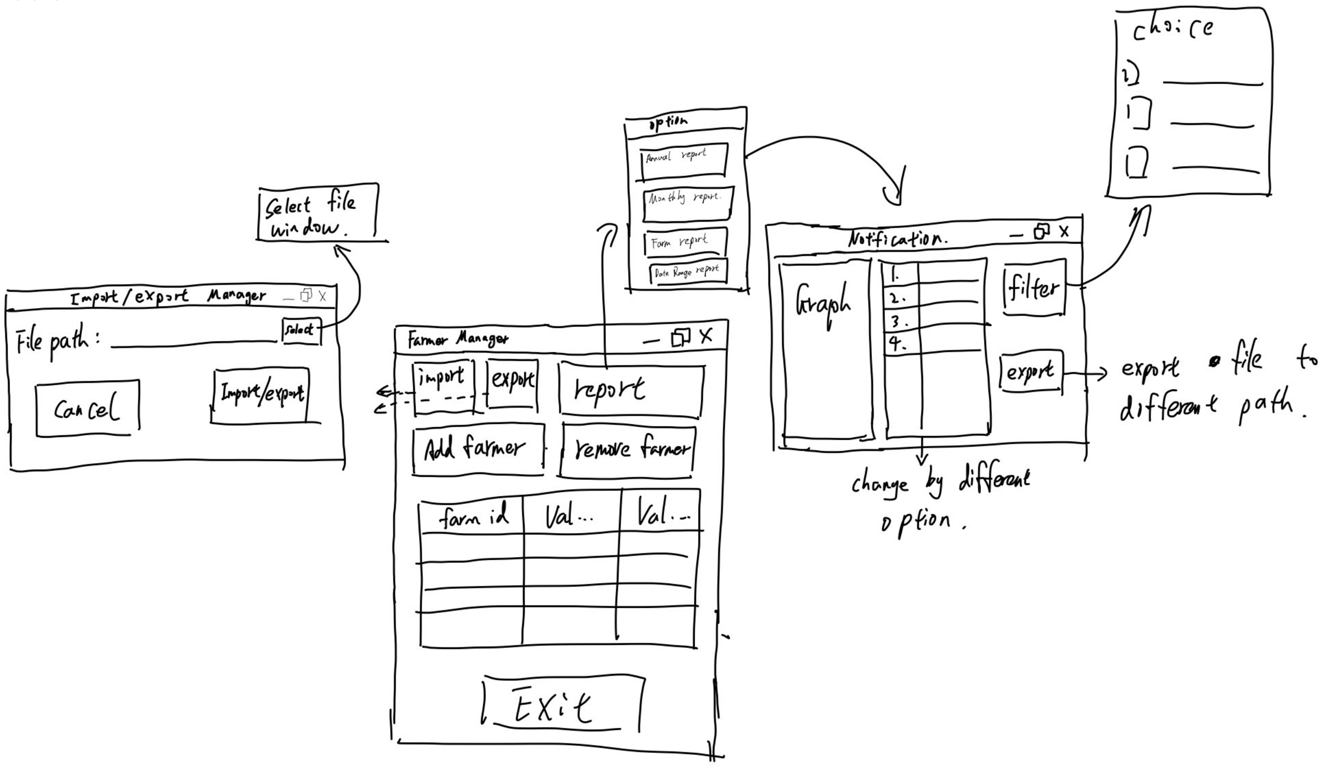
7 Procedure

This part will describe the different possible running cases.

I think this part is useless right now. Each part before has been discussed throughly. I will discuss a simple running below:

User start the program, Main window will be shown up with empty left table. Then click import(you will get an empty file if you try to export now) to import files. Import path could be single file or list of files. After that, they could click report to get the report window, each button will generate a specific report. In each report window, they could filter data. The filter result or analyzation result could be exported by using the export function.

GUI looks like below:


Select file window.
Import/export Manager
File path :
select
Cancel
Import/export
Farmer Manager
import
export
report
Add farmer
remove farmer
farm id
Val...
Val...
Exit
option
Annual report
Monthly report.
Form report
Date Range report
Notification.
Graph
1.
2.
3.
4.
filter
export
export file to different path.
change by different option.
choice
1)