Final Project
Electricity consumption tracking with GUI

Written by: Quang Luong – e1500949

Group No.: I-IT-4N1

Submit date: 16/03/2019

I. Introduction

The user can give consumption values of a house (or different electric devices) and the program draws a bar chart of the consumption data. The program tracks consumption by every month in a year. The program also calculates the total consumption, average consumption, and finds minimum and maximum of the consumption data. The current year will be automatically taken from system calendar.

The program also gives user a function to save consumption data to a file, so that when next time user starts the program, he or she can open the file and add more data.

II. Implementation

1. Initial class

This is a GUI program; therefore, the whole program code should be put inside a class, which is extended from Application.

```
public class ElectricityConsumption extends Application {
    @Override
    public void start(Stage stage) {

    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

The program also uses the bar chart for display data of consumption, File Stream for read from file and store data to file, so, we have to declare them at the beginning of the class. It should be global variable for re-using in multiple methods later.

```
private final FileChooser fileChooser = new FileChooser();

final CategoryAxis xAxis = new CategoryAxis(); //String category
final NumberAxis yAxis = new NumberAxis();
final BarChart<String, Number> barchart = new BarChart<>(xAxis,yAxis);
XYChart.Series<String, Number> series1 = new Series<>();
```

Here we use Tree Map for store data, because of its sortable characteristic. With this, we can store the consumption data in pair of key=value exactly like HashMap and we also can sort it in order of months in a year.

```
Map<String, Double> consumptionList = new TreeMap<String, Double>();
```

We also define some global variables for statistical data

```
    Double sum = 0.0;
    Double average = 0.0;
    Double min = 9999.0;
    Double max = 0.0;
    int count = 0;
    // array list to store value for finding min and max values
    ArrayList<Double> al = new ArrayList<Double>();
```

2. Stage setup

At the beginning of method **start(Stage stage),** we define and setup everything for our program UI. Here we use a combination of Grid pane and Border pane for the UI.

```
        stage.setTitle("Electricity consumption tracking");

        BorderPane mainpanel = new BorderPane();

        GridPane grid = new GridPane();
        grid.setAlignment(Pos.TOP_CENTER);
        grid.setHgap(10); //Horizontal space between columns
        grid.setVgap(10); //Vertical space between rows
```

We also need a Menu bar with option New, Open, Save and Exit to control the program.

```
        MenuBar menuBar = new MenuBar();
        Menu menuFile = new Menu("File");
        menuBar.getMenus().add(menuFile);

        MenuItem newFile = new MenuItem("New");
        MenuItem open = new MenuItem("Open");
        MenuItem save = new MenuItem("Save");
        MenuItem exit = new MenuItem("Exit");
        menuFile.getItems().addAll(newFile, open, save, new
                                   SeparatorMenuItem(),  exit);
```

3. Reuse methods

In our program, we have some code blocks which are executed many times, so we can extract them as method for re-use. Here are those method.

First, we need the chart update dynamically, so we define a method for it.

```
private void drawChart() {
        consumptionList.remove("year"); //make sure the chart contains only
                                          months, no more year value once it
                                          is drawn
        consumptionList.entrySet().forEach(
          (HashMap.Entry<String, Double> entry) -> {
            String tmpString = entry.getKey();
            Number tmpValue = entry.getValue();
            series1.getData().add(new Data<>(tmpString, tmpValue));
        });
    }
```

We also need the statistical data update every times user enter a new value. But before update, we need to clear old data first

```
private void resetInfo() {
        min = 9999.0;
        max = 0.0;
        average = 0.0;
        count = 0;
        sum=0.0;
        lbl_min.setText("Min: ");
        lbl_max.setText("Max: ");
        lbl_total.setText("Total:");
        lbl_avg.setText("Average: ");
    }
```

```
private void getInfo() {
        resetInfo();
        al.clear();
        lbl_min.setText("Min: " + getMin().toString() + " kWh");
        lbl_max.setText("Max: " + getMax().toString() + " kWh");
        lbl_total.setText("Total: " + getTotal().toString() + " kWh");
        lbl_avg.setText("Average: " + getAverage().toString() + " kWh");
    }
```

4. Functions

a. Get the year from system calendar

The program will automatically get the year value from system calendar with the following code

```
Double currentYear = (double) Calendar.getInstance().get(Calendar.YEAR);
```

The reason that we cast year to double is for storing to the Map<String, Double> consumptionList when saving it to file later.

b. Get statistical data

Those values such as Total, Average, Minimum, Maximum can be found with these method in our program

We loop through every value in the list and get the total by add one by one value

```
private Double getTotal() {
        sum=0.0;
        consumptionList.values().forEach(value -> {
            sum += value;
        });
        return sum;
    }
```

Then we use that sum value to calculate the average, but before that, we should loop through list again and count how many month in the list has value

```
private Double getAverage() {
        count = 0;
        for (Double m : consumptionList.values()) {
            if(m != 0) count++;
        }
        average = Math.round(sum / count* 10) / 10.0;    //Get only 1 decimal
                                                           number
        return average;
    }
```

With the minimum and maximum value, we operate them with the same theory algorithm, add all value from Tree Map to a temp array list, then loop through that list to find min. and max. values.

```
private Double getMax() {
        for (Double m : consumptionList.values()) {
            al.add(m);
        }
        for (int i=0; i<al.size();i++) {
            if (al.get(i)> max) {
                max = al.get(i);
            }
        }
        return max;
    }

    private Double getMin() {
        for (Double m : consumptionList.values()) {
            al.add(m);
        }
        for (int i=0; i<al.size();i++) {
            if (al.get(i)< min && al.get(i) != 0) {
                min = al.get(i);
            }
        }
        return min;
    }
```

c. Insert data

When user do not enter any input or do not select the month, the program will not allow user to insert the data, the insert button will be disable in this case. This can be done by following code with disableProperty() and bind() method

```
//check if user input value and select month
        insert.disableProperty().bind(
                    txtF_Cons.textProperty().isEmpty()
                .or(cmb_Month.valueProperty().isNull())
        );
```

When user enter for both field Consumption and Month then click Insert button, the following code will be executed with EventHandler method.

```
insert.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent arg0) {
                try {
                    double cons = Double.parseDouble(txtF_Cons.getText());
                    String month = cmb_Month.getValue();
                    consumptionList.put(month, cons);
                    drawChart();
                    getInfo();
                    afterClick();
                } catch (NumberFormatException e) {
                    Alert alert = new Alert(AlertType.ERROR);
                    alert.setHeaderText("Check your input!");
                    alert.setContentText("Enter a number value.");
                    afterClick();
                    alert.showAndWait();
                }
            }
            public void afterClick() {
                txtF_Cons.clear();
                txtF_Cons.requestFocus();
                cmb_Month.getSelectionModel().clearSelection();
            }
        });
```

We want to make sure that user must enter a real and true format value, so we put the code in a try-catch block to check if consumption value is a number. If not, the program will alert with a custom alert to user.

d. Create new set data

With option New from menu file, the program will clear everything from stage.

```
newFile.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent arg0) {
                txtF_Cons.clear();
                txtF_Cons.requestFocus();
                cmb_Month.getSelectionModel().clearSelection();
                resetInfo();
                resetChart();
                drawChart();
            }
        });
```

e. Save data to File

The data should be saved to a file, so next time user start the program, he or she can open that file and add more data for next months. We want to let user chooses the location where they want to store their file, therefore, every times user clicks on Save option in Menu bar, the program will show a pop-up window let user choose the location.

```
save.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                 File file = fileChooser.showSaveDialog(stage);
                 if (file != null) {
                    saveConsToFile(file);
                 }
            }
        });
```

When user choose the location and input file name, the program will save all the data from consumptionList Tree Map to a file as an object. Of course, we also want to store the year of that data to a file too, so we have to put year value to the list before program write it to the file.

```
private void saveConsToFile(File file) {
        System.out.println(file.getAbsolutePath());
        consumptionList.put("year", currentYear);   // add year to TreeMap for
                                                       using when open it
                                                       later
        try(ObjectOutputStream output = new ObjectOutputStream(new
                                                FileOutputStream(file))){
            output.writeObject(consumptionList);
        }
        catch(IOException ex){
            System.out.println("Error: " + ex.getMessage());
        }
    }
```

f. Open and read data from file

When user one to open a store file, they can do by choosing Open from menu bar. The program will let user choose which file should be open for use.

```
save.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                 File file = fileChooser.showSaveDialog(stage);
                 if (file != null) {
                    saveConsToFile(file);
                 }
            }
        });
```

Once a file is selected, the program will first clear everything from previous task then read data saved from that file and populate to corresponding variable. Our program will store the object of Tree Map saved from file and put it back to consumptionList for drawing chart. The year value also gets from the file and set to title of the chart. Then the program draws bar chart based on value get from file and also calculate statistical data.

```
private void readConsFromFile(File file) throws ClassNotFoundException{
        resetChart();
        drawChart();
        try(ObjectInputStream input = new ObjectInputStream(new
                                                FileInputStream(file))){
            consumptionList = (Map<String, Double>) input.readObject();
            currentYear = consumptionList.get("year");
            barchart.setTitle("Year " + currentYear.intValue());
            drawChart();
            getInfo();

        }
        catch(IOException ex){
            System.out.println("Error: " + ex.getMessage());
        }
    }
```

g. Exit option

Here we also include another option for user close the program, user can choose the Exit option from the menu bar, the program will close the stage once user chooses this action.

```
exit.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                Platform.exit();
            }
        });
```

III. Test run

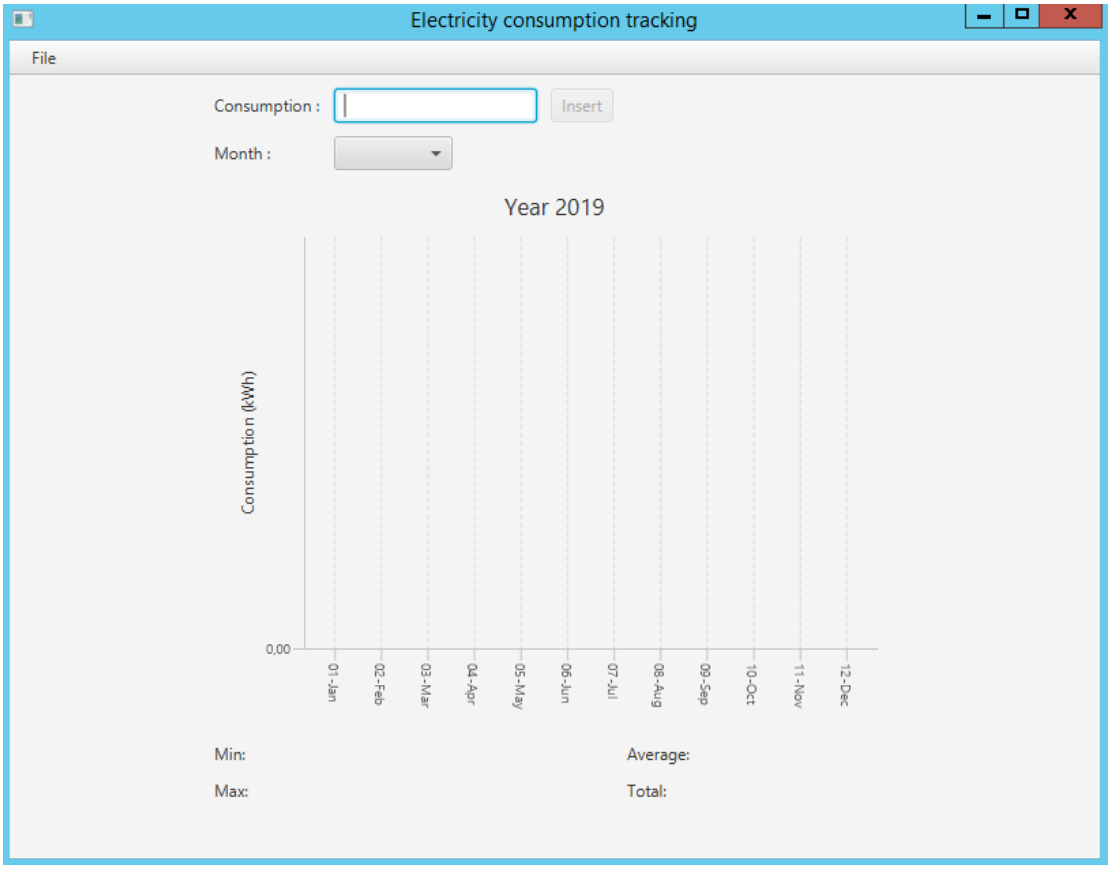

Figure 1. Program GUI

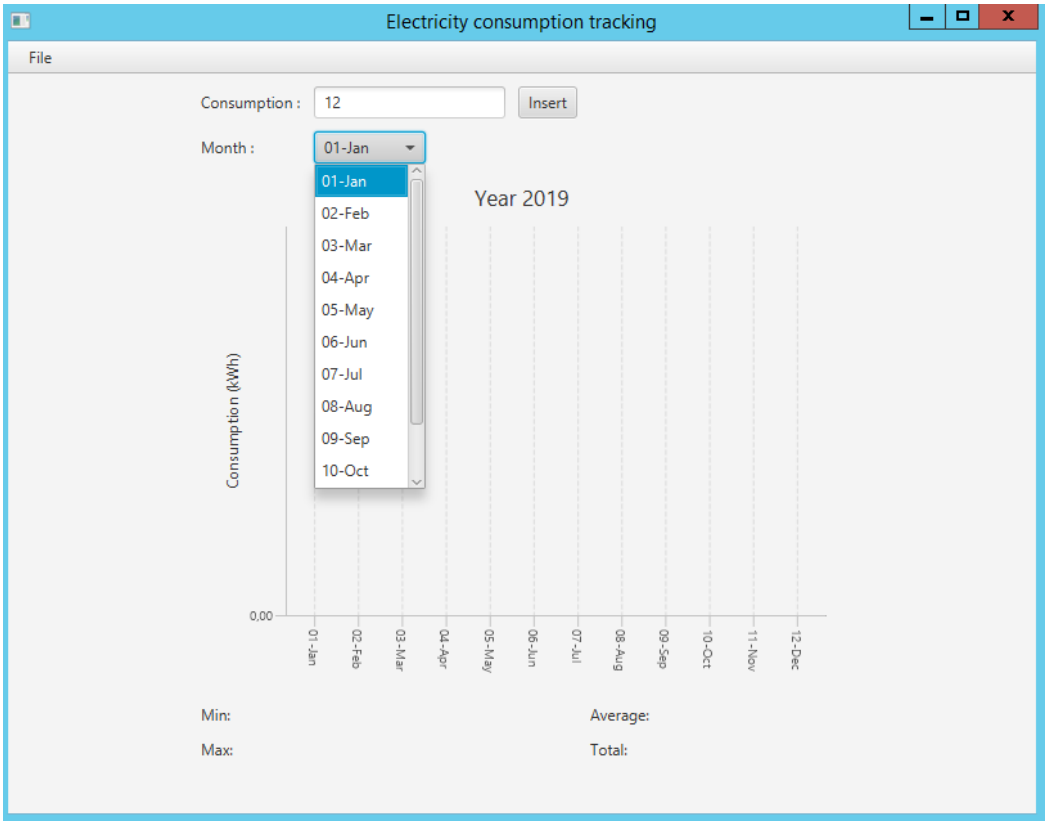

Figure 2. List of Month in combo box for user selection

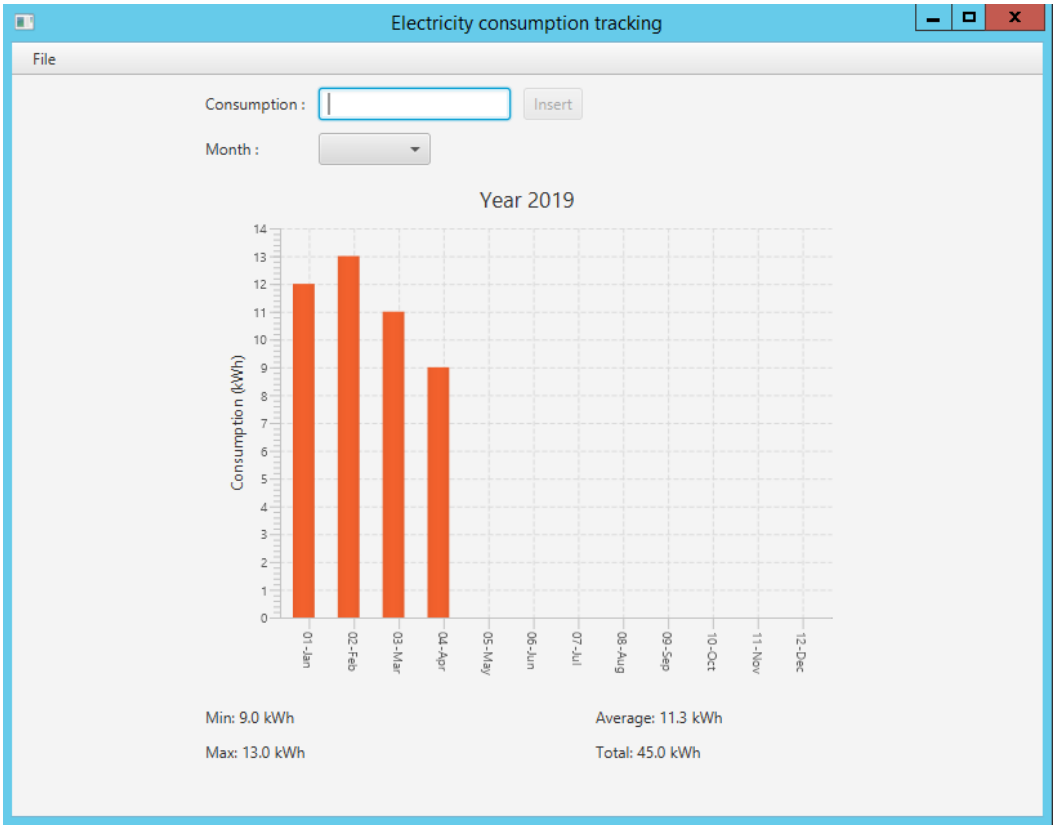

Figure 3. Some data is added, the program worked well

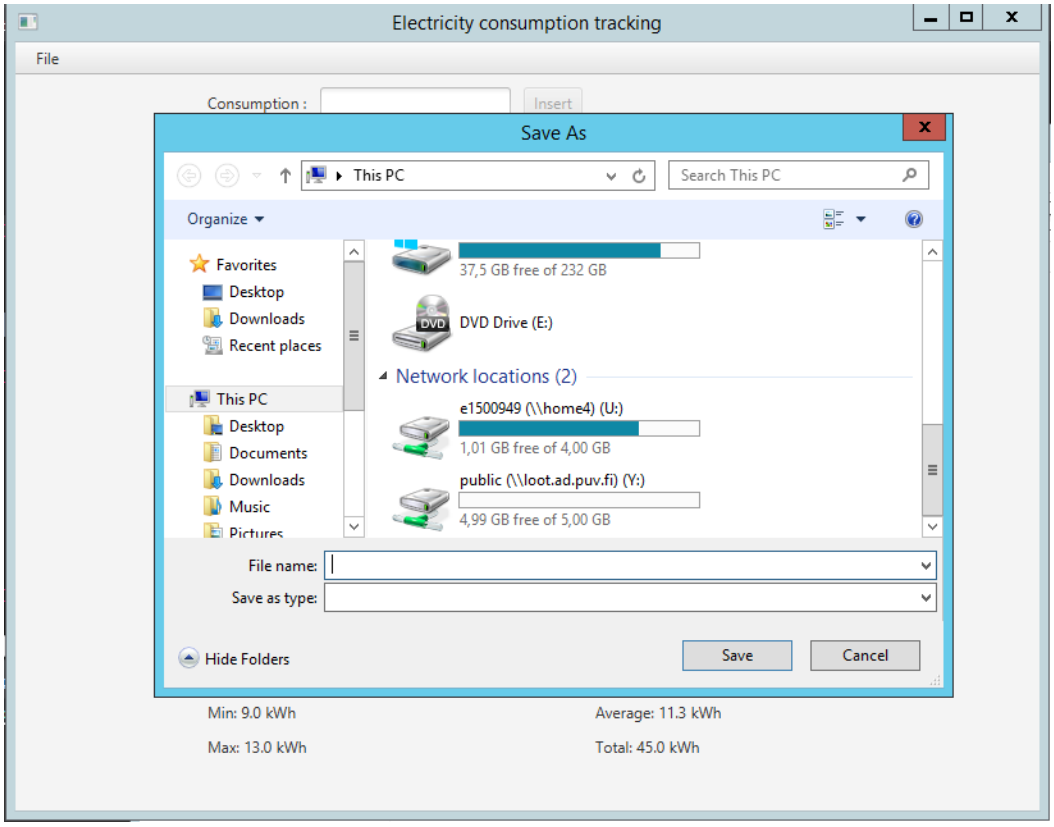

Figure 4. Save file dialog

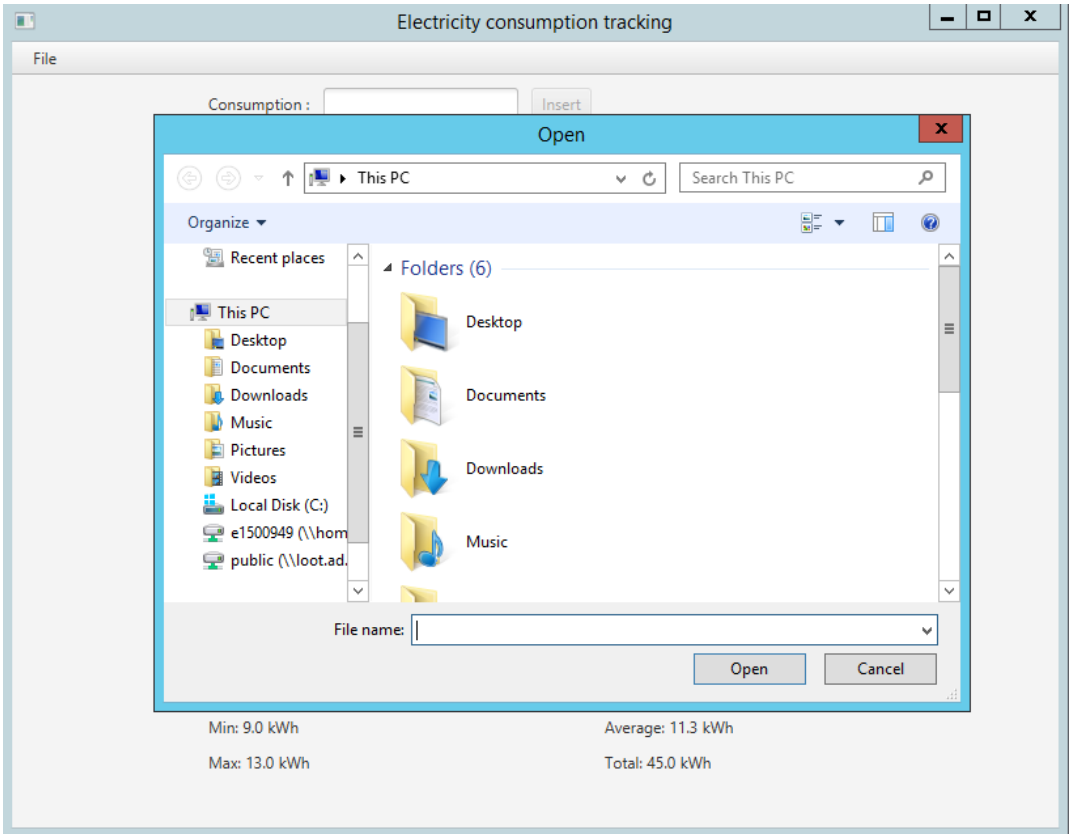

Figure 5. Open file dialog

We also create some test files with these values for testing the program

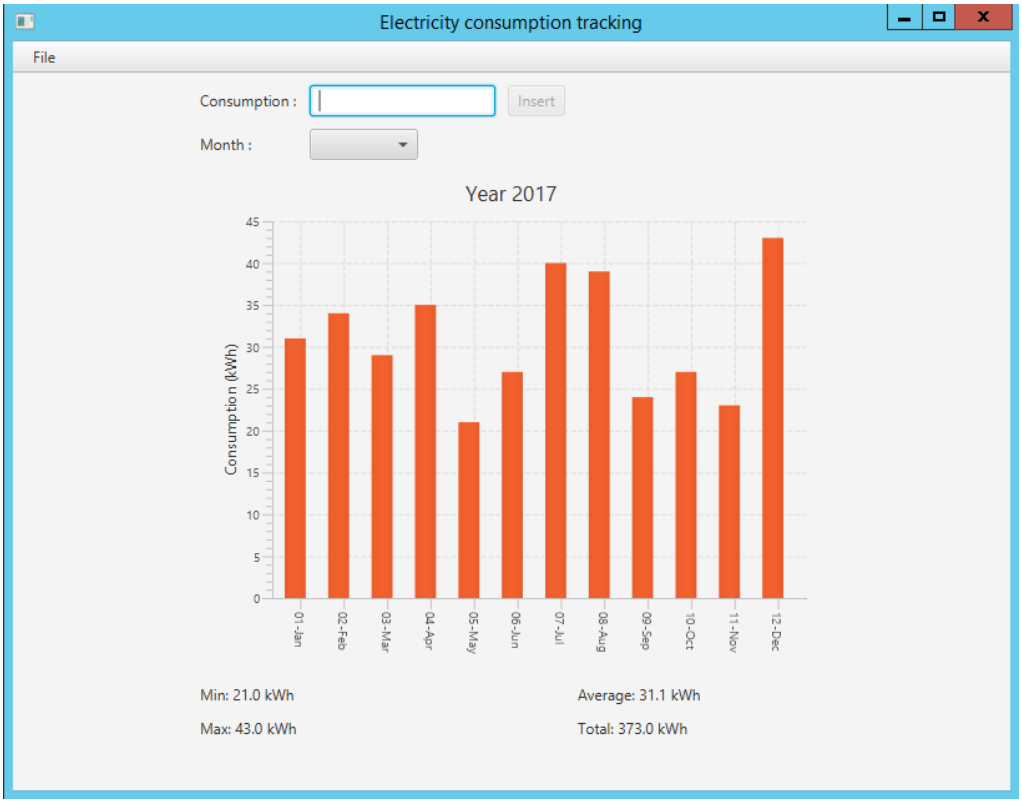

Figure 6. Consumption data read from file of year 2017

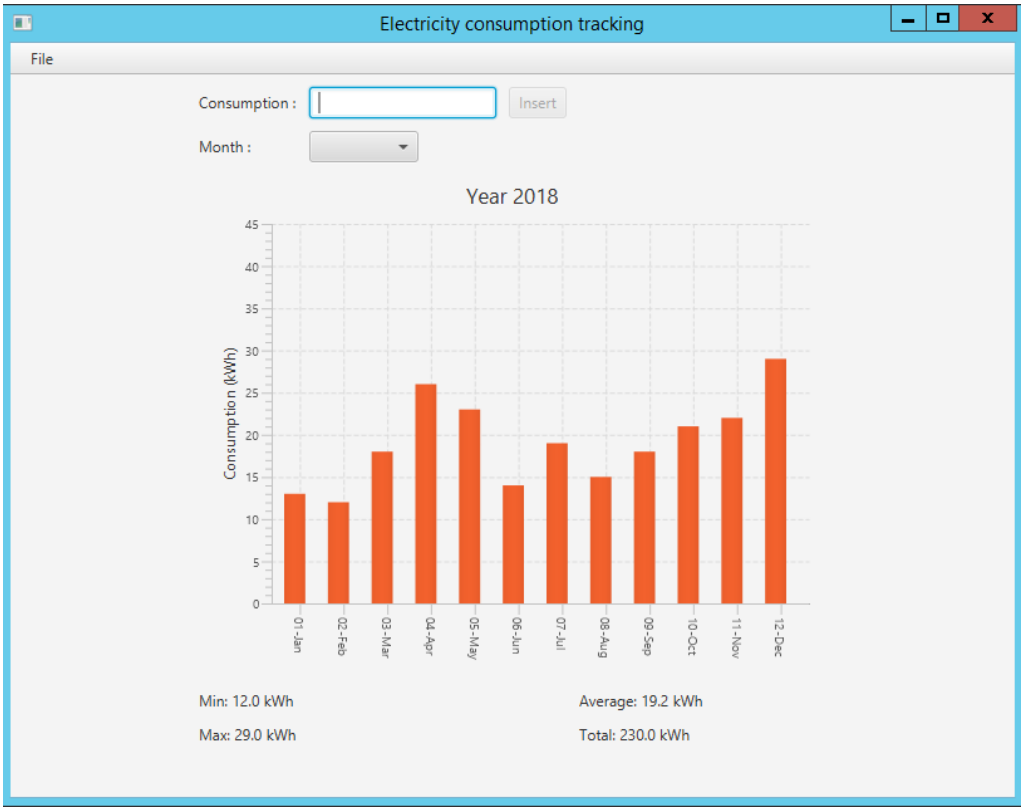

Figure 7. Consumption data read from file of year 2018

IV. Deployment

In this section, we do follow steps to create an executable JAR file from our program class. The JAR file can be exported by eclipse built-in feature.

- From the menu bar's **File menu**, select **Export**.
- Expand the **Java** node and select **Runnable JAR file**. Click **Next**.
- In the Runnable JAR File Specification page, select our program launch configuration to use to create a runnable JAR. In this case is ***ElectricityConsumption - finalProject***
- In the **Export** destination field, either type or click **Browse** to select a location for the JAR file. We can save it within the project folder.
- Select an appropriate library handling strategy. We should choose the second one.
- Then click on **Finish**

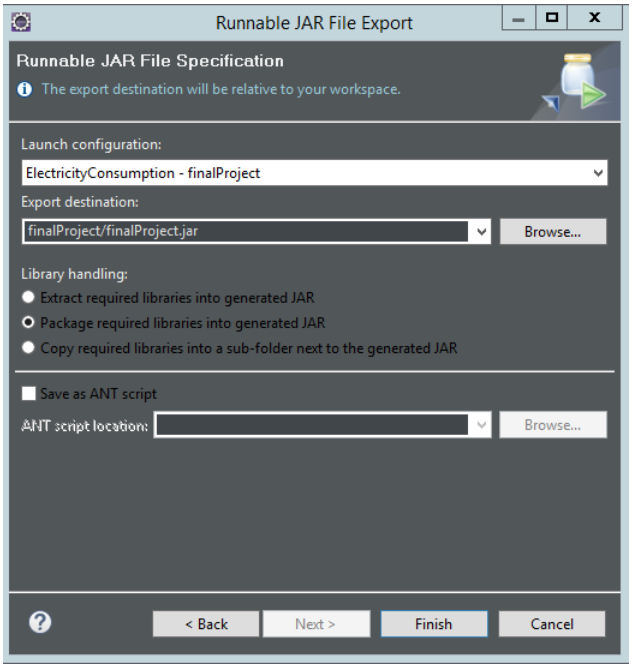

Figure 8. Export option of JAR file

V. SUMMARY

This program is just a simple program, which contain enough basic functions for tracking electricity consumption. In future, if we have a chance to work with this project again, we can update it with more features, so that user can easily use this program in their life without inconvenient.