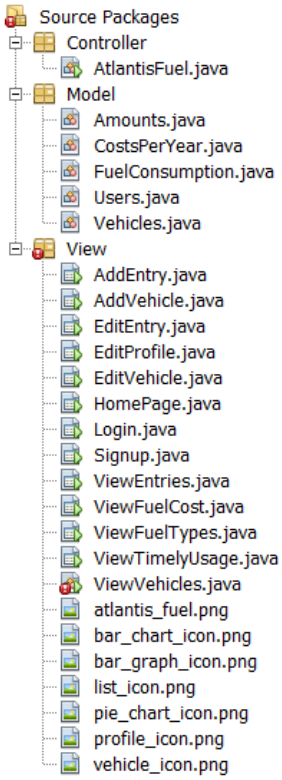
A client consultation was conducted (see Appendix 1.2) prior to the development phase outlined in this document.

Product Structure



For this solution, Model View Controller Package design was used, as in Figure 1. Firstly, the Model package, controlled and stored data, including the encapsulation of variables and data structures. The View package controlled user interactions with GUIs and initiated algorithms from the controller according to user demands. The Controller package controlled contained methods and data structures used widely within the program. The main advantages of this setup are listed below.

- Modular programming which is the division of the entire program into manageable tasks
  - Effective maintenance/modification and addressal of issues/bugs due to easier identification
  - Manageable tasks (can be split up, programmed concurrently, more accurate addressal of each task)
  - Effective program readability (code reusability, shared methods in main class (AtlantisFuel (see Figure 1))
  - Easy planning

Figure 1: MVC Package Design for Atlantis Fuel (Model, View, Controller)

Table of Tools

Tool/Technique	Implementation
ArrayList	Used to store a user, fuel, vehicle data etc.
Encapsulation	Ensured data in ArrayLists was secured and not accidentally modified.
Default Table Model	This tool was used to store users' individual fuel data and present it in rows and columns in the Table of Fuel Entries.
Serialization	Used to store user, fuel and vehicle data in a volatile manner, through byte streams.
Deserialization	Used to recover serialized information and convert it to a comprehensible format for user.
Type Casting	Convert between data types e.g., fuel costs were inputted as strings but converted to floats .
Bar chart	Used to present fuel data in the ViewFuelCost and ViewTimelyUsage classes.
JTable	Presented fuel data to users as a table, constructed in conjunction with Default Table Model.
Comparator	Used for sorting values within ArrayLists.

Code Libraries Used

Library	Technique
iText Library	Used to format the PDF document containing fuel data as will be discussed below.
JFreeChart Library	Allowed the creation of a chart, which displayed graphs.
JCommon Library	Used to edit text and labels of graphs.

Bar chart

A bar chart was used to display user's annual fuel usage. First, the user's data was retrieved from the fuelConsumption ArrayList through a loop and added to a new encapsulated ArrayList, amounts, temporarily storing the user's data of fuel amounts and the years purchased. The amounts ArrayList was instantiated, as in Figure 2.1, and was also cleared at the beginning of every instance of the class so the data did not collide with previous versions of itself.

```
public static ArrayList<Amounts> amounts = new ArrayList<>();
/*amounts arraylist encapsulated in Amounts class in model.
this arraylist will store small versions of the user's fuel objects
i.e. only the year of the fuel bought (year) and the amount of fuel (amount).
*/
```

Figure 2.1: Instantiated amounts ArrayList

The findYears() method for transferring data between ArrayLists fuelConsumption and amounts is shown in Figure 2.2. This method looped through all fuel entries and collected the year and amount of any fuel entries in the ArrayList belonging to the current user.

```
public void findYears(){
    try {
        AtlantisFuel.deserialiseFuel();
        /*fuelConsumption arraylist must be deserialized to access its data*/
    } catch (IOException ex) {
        Logger.getLogger(ViewTimelyUsage.class.getName()).log(Level.SEVERE, null, ex);
    }

    for(int k = 0; k < AtlantisFuel.fuelConsumption.size(); k++){
        if(AtlantisFuel.fuelConsumption.get(k).getUsername().equals(HomePage.username)){
        /*if the entry currently being looped through has the same username as
        the logged in user then it's data must be stored to add to the bar chart*/
            int j = 0;
            while(found == false && j < amounts.size()){
            /*this method loops through the amounts arraylist to find out whether the
            year has already been added to the arraylist and the amount simply needs to be incremented
            or whether there are no previous entries for this year and thus it needs to be added to a
            new index in the amounts arraylist*/
                if(amounts.get(j).getYear().equals((String.valueOf(
                        AtlantisFuel.fuelConsumption.get(k).getDateEnterred().getYear() + 1900)))){
                /*this if statement checks the aforementioned condition for a year already being in the
                amounts arraylist*/
                    int currentAmount = amounts.get(j).getAmount();/*the current fuel amount is stored*/
                    amounts.get(j).setAmount(currentAmount + AtlantisFuel.fuelConsumption.get(k).getFuelAmount());
                    /*the amount is updated as the previously stored amount + the new amount from the current entry*/
                    found = true;
                }
                else{
                    j = j + 1;
                }
            }
```

Figure 2.2: First half of findYears() method – Creates an ArrayList storing only the current user's data.

The code above used a for loop, while loop and conditional statements to check every entry entered and find those pertaining to the current user. As commented, it either increments the amount of fuel already purchased for a particular year or instantiates another index in the amounts ArrayList to store a counter of fuel amount for a new year that is not yet listed in the ArrayList. Hence, each index in the ArrayList will correspond to one bar in the bar chart, representing fuel usage in a specific year.

```
DefaultCategoryDataset ds = new DefaultCategoryDataset();
    ds.clear(); //removes data that may have been previously stored
for(int g = 0; g < amounts.size(); g++){
    ds.setValue(amounts.get(g).getAmount(), "Litres", amounts.get(g).getYear());
}
//instantiating chart - assigning title, axis labels, orientation of the bars
JFreeChart chart = ChartFactory.createBarChart("Usage across the Years", "Year", "Litres", ds,
                                                PlotOrientation.VERTICAL, true, true, false);
CategoryPlot plot = chart.getCategoryPlot();
plot.setRangeGridlinePaint(Color.black);
ChartPanel panel = new ChartPanel(chart); //adds the chart to the full chart panel
jPanel1.removeAll(); //removes any previous bar chart
jPanel1.add(panel); //adds the full chart panel to the panel of the jFrame
jPanel1.setLayout(new BorderLayout());
jPanel1.add(panel, BorderLayout.NORTH);
jPanel1.updateUI();
```

Figure 2.3: Second Half of findYears() method – Creates a Data Set with data from amounts ArrayList

The second section of the findYears() method, see Figure 2.3, outlines the creation of the data set which was displayed in a bar graph. The JFreeChart library was imported to access methods to load data sets into graphs. The for loop added each index of the amounts ArrayList to the data set, ds. Then the chart was instantiated, formatted, and finally added to a JPanel on the JFrame. The dataset and panel were also cleared before displaying the chart to avoid collisions with any previously stored data. The finished display of this is in Figure 2.4.

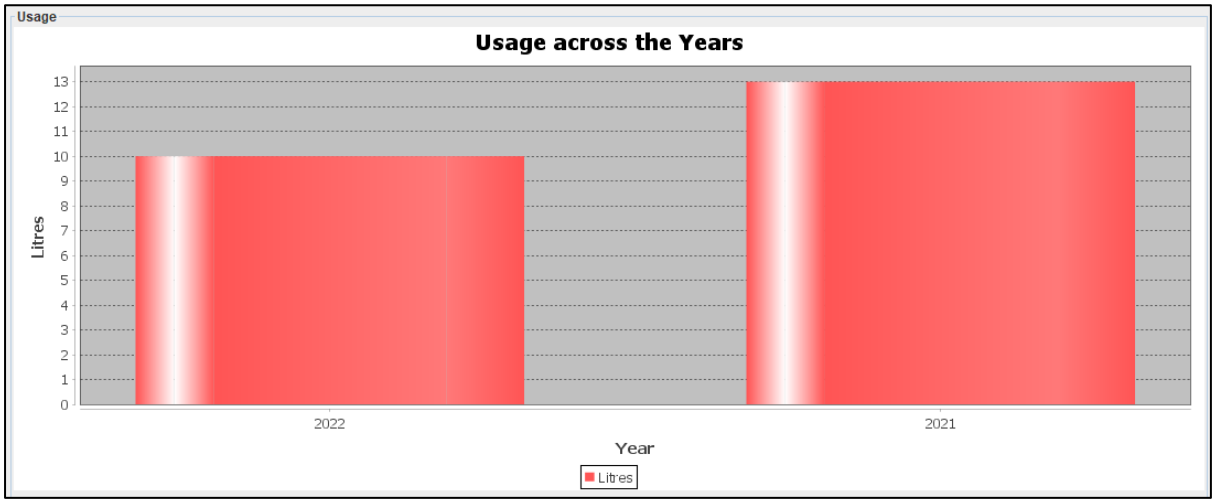


Figure 2.4: ViewTimelyUsage Graph Showing Fuel Usage Per Year

The technique was highly ingenious as Figure 2.2 and 2.3 were created distinctly to transfer ArrayList data into a data set to satisfy the client's criteria. The method also had high complexity as there were precisely nested loops and conditional statements to ensure successful execution whilst considering possible issues. Thus, success criteria 7 was satisfied, allowing users to view graphs of fuel data. Alternatively, a linked list was considered to store

the temporary data since it has faster traversal however it did not allow insertion into existing indices, which was necessary to increment the fuel amounts.

Word Count (1): 399

Sorting Table of Fuel Entries

The ViewEntries class allowed the client to view all their existing entries and download a PDF File containing all their fuel entries sorted in their chosen manner. The options offered to the user are shown in Figure 3.1.

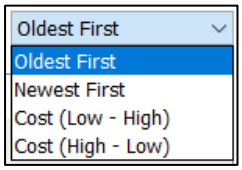


Figure 3.1: Options Presented to User for Sorting Fuel Data Entries in Table.

An action event tool, see Figure 3.2, triggered code to execute the user's selection in Figure 3.1. Thus, if the user changed their preference, the code under the Action Event would execute. This has low complexity.

```
/*the following ActionEvent will sort the table as requested by the user once
an option is clicked from the drop down menu*/
private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
```

Figure 3.2: Action Event Method for Sorting Table of Entries.

Following this, the fuelConsumption ArrayList was deserialized to access the data. A try-catch statement, in Figure 3.3, was used to avoid possible errors that would crash the application. An if-else statement could have been used to list all possible errors however this would be less elegant than a try-catch statement which addresses all errors.

```
try { /*try-catch statement avoids program crashing if errors occur in the midst
    of deserializing the arraylist.*/
    AtlantisFuel.deserialiseFuel(); /*gets the method from main class*/
} catch (IOException ex) {
    Logger.getLogger(ViewEntries.class.getName()).log(Level.SEVERE, null, ex);
}
```

Figure 3.3: Try-catch Statement for Deserialization

The code within the Action Event has been shown below in Figure 3.4. Multiple if-else statements were used to find the option the user chose from Figure 3.1. Then the fuelConsumption ArrayList was sorted using a Comparator function by either cost or date. To sort the ArrayList by "Newest First", the ArrayList was first sorted in the order "Oldest First" and then reversed since no unique method existed. A similar process was used to sort by "Cost (High - Low)". Then the ArrayList was serialized to save the updated order and the table of entries was re-created to show the re-ordered data. The method for the addition of data to the table was a separate technique and located in the main class, AtlantisFuel.

```
if(jComboBox1.getSelectedItem().equals("Newest First")){
    Collections.sort(AtlantisFuel.fuelConsumption, Comparator.comparing(FuelConsumption::getDateEnterred));
    Collections.reverse(AtlantisFuel.fuelConsumption);
} else if(jComboBox1.getSelectedItem().equals("Oldest First")){
    Collections.sort(AtlantisFuel.fuelConsumption, Comparator.comparing(FuelConsumption::getDateEnterred));
} else if(jComboBox1.getSelectedItem().equals("Cost (Low - High)")){
    Collections.sort(AtlantisFuel.fuelConsumption, Comparator.comparing(FuelConsumption::getCost));
} else if(jComboBox1.getSelectedItem().equals("Cost (High - Low)")){
    Collections.sort(AtlantisFuel.fuelConsumption, Comparator.comparing(FuelConsumption::getCost));
    Collections.reverse(AtlantisFuel.fuelConsumption);
}
AtlantisFuel.serialiseFuel(); /*serialize arraylist to save updates made*/
AtlantisFuel.addRowToJTable(); /*re-creates the table of entries using the updates ArrayList*/
```

Figure 3.4: Code Executed to Sort Table of Entries.

An example result of the execution of the above code is shown below. This process satisfied success criterion 5, allowing users to view and sort the data entries. This method contained high ingenuity as it combined many tools to achieve the success criterion.

Cost (Low - High)

ID Number	Day	Month	Year	Type	Amount	Cost
2	01	4	2022	E10	7	7.0
1	01	1	2021	E10	20	40.0
4	06	9	2021	Diesel	42	69.0
3	31	12	2021	E10	88	88.0

Figure 3.5: Table of Entries Ordered by Ascending Cost.

Word Count (2): 294

PDF Download Option

The “DOWNLOAD DATA” JButton in the ViewEntries GUI allowed users to download a PDF of their fuel entries to their device. The Action Event in Figure 4.1 was used to recognise when the user clicked the button.

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    /*when user clicks DOWNLOAD DATA this code runs. The following method
    runs which formats and downloads the PDF*/
    downloadPDF();
}
```

Figure 4.1: Action Event for “DOWNLOAD DATA” Button

As seen above this activated the downloadPDF() method which downloaded a PDF titled “AtlantisFuel.pdf” to the user’s devices’ downloads folder. The method first instantiated the document and found the user’s home file location, as in Figure 4.2. It then used the File OutputStream to assign the file’s final location to the user’s downloads folder. This was done within a try-catch statement to once again prevent the program crashing due to being unable to find the user’s downloads folder.

```
String home = System.getProperty("user.home"); //locates address of user's home file location
Document document = new Document();
//the itext library has been imported to enable the following code

try {
    PdfWriter.getInstance(document, new FileOutputStream(home+"/Downloads/"+"AtlantisFuel.pdf"));
```

Figure 4.2: downloadPDF() method Part 1

In Figure 4.3, the title of the document was assigned using the Chapters tool. A table was created within the PDF with six columns to display the fuel entries. A for loop iterated through the table in the GUI itself and stored the values as Strings which were then transferred into cells in the table. Finally, the table was added to the document.

```
//adding a title to the pdf
int currentUser = AtlantisFuel.findUser(HomePage.username); //retrieves username
Paragraph chapterTitle = new Paragraph(AtlantisFuel.users.get(currentUser).getFirstname()+"'s Fuel Usage\n"
        + " ");
Chapter chapter1 = new Chapter(chapterTitle, 1); /*instantiates new title, will require one line on the pdf*/
chapter1.setNumberDepth(0);
document.add(chapter1); //adding the title to the document

//6 columns required in this table to match headers of table in jframe
PdfPTable table = new PdfPTable(6);
table.addCell("Day");
table.addCell("Month");
table.addCell("Year");
table.addCell("Type");
table.addCell("Amount");
table.addCell("Cost");

/*loops for number of rows in table, loads all values into separate strings which will be
added as cells in the table in the pdf*/
for(int i = 0; i < jTable1.getRowCount(); i++){
    String d = jTable1.getValueAt(i,1).toString();
    String m = jTable1.getValueAt(i, 2).toString();
    String y = jTable1.getValueAt(i, 3).toString();
    String t = jTable1.getValueAt(i, 4).toString();
    String a = jTable1.getValueAt(i, 5).toString();
    String c = jTable1.getValueAt(i, 6).toString();

    //data added to table in document as strings
    table.addCell(d);
    table.addCell(m);
    table.addCell(y);
    table.addCell(t);
    table.addCell(a);
    table.addCell(c);
}

document.add(table);
```

Figure 4.3: downloadPDF() method Part 2; formatting a table within the PDF

The outcome of this can be seen in Figure 4.4, which corresponds to the entries in Figure 3.5.

Person's Fuel Usage

Day	Month	Year	Type	Amount	Cost
01	4	2022	E10	7	7.0
01	1	2021	E10	20	40.0
06	9	2021	Diesel	42	69.0
31	12	2021	E10	88	88.0

Figure 4.4: Table within Downloaded PDF

This process satisfied success criterion 6, which outlined that users can download a PDF of the fuel data. This process contained high complexity and ingenuity as the tools were manipulated uniquely to fit the data and requirements. Using FileWriter over File OutputStream was

considered however File OutputStream was used throughout the program e.g., for serialization, it was chosen to maintain cohesion throughout the code.

Word Count (3): 260

Word Count (Total Criterion C): 1024

References

1bestCsharp Blog. (2022). *Java - Populate JTable From ArrayList In Java*. Tutorials. https://1bestcsharp.blogspot.com/2016/03/java-populate-jtable-from-arraylist.html

ClipArtMax. (2019a). 049 Checklist 2 Icon - Check List Svg Icon [Online Image]. In *ClipArtMax*. https://www.clipartmax.com/middle/m2i8N4d3A0b1i8N4_049-checklist-2-icon-check-list-svg-icon/

ClipArtMax. (2019b). Awning Replacement - Recreational Vehicle [Online Image]. In *ClipArtMax*. https://www.clipartmax.com/middle/m2i8i8N4Z5G6b1i8_awning-replacement-recreational-vehicle/

ClipArtMax. (2019c). Bar Chart, Bar Chart, Business Graph Icon - Histogram Cartoon [Online Image]. In *ClipArtMax*. https://www.clipartmax.com/middle/m2i8m2K9Z5G6m2d3_bar-chart-bar-chart-business-graph-icon-histogram-cartoon/

ClipArtMax. (2019d). Bar Chart Free Icon - Loss Graph Png [Onine Image]. In *ClipArtMax*. https://www.clipartmax.com/middle/m2i8d3i8G6m2i8i8_bar-chart-free-icon-loss-graph-png/

ClipArtMax. (2019e). Clipart Info - Green Paper Airplane Icon [Online Image]. In *ClipArtMax*. https://www.clipartmax.com/middle/m2i8H7A0b1K9N4N4_clipart-info-green-paper-airplane-icon/

ClipArtMax. (2019f). Pie Chart Icon Flat [Online Image]. In *ClipArtMax*. https://www.clipartmax.com/middle/m2H7N4d3K9b1Z5m2_512-x-512-4-pie-chart-icon-flat/

CodeTech Club. (2015). How to make bar chart in java using JFreeChart [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=RUDrjqywD1g

Create JFrame and coponents using code. (n.d.). [YouTube Video]. In *YouTube*. Retrieved March 13, 2022, from https://www.youtube.com/watch?v=PUeiDFWreFc

Developer. (2017). Generate pdf in java [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=TAJoVkaVv-8

Hex to RGB Color Converter. (n.d.). Www.rapidtables.com. Retrieved March 13, 2022, from https://www.rapidtables.com/convert/color/hex-to-rgb.html

JavaTPoint. (2021). *Java Date getYear() Method*. JavaTPoint. https://www.javatpoint.com/java-date-getyear-method

Liss, A. (2015, May 24). *Java String - See if a string contains only numbers and not letters*. Stack Overflow. https://stackoverflow.com/questions/10575624/java-string-see-if-a-string-contains-only-numbers-and-not-letters

Krishan. (2016, July 13). *java - How to delete all components in a JPanel dynamically*. Stack Overflow. https://stackoverflow.com/questions/38349445/how-to-delete-all-components-in-a-jpanel-dynamically

Ranschaert, B. (2019, February 25). *What is the best way to find the user's home directory in Java?* Stack Overflow. https://stackoverflow.com/questions/585534/what-is-the-best-way-to-find-the-users-home-directory-in-java

bezkoder. (2019, December 27). *Java - Sort ArrayList of Objects*. BezKoder. https://www.bezkoder.com/java-sort-arraylist-of-objects/

Jenkov, J. (2021, March 9). *Java SimpleDateFormat*. JENKOV.com. https://jenkov.com/tutorials/java-internationalization/simpledateformat.html

Download itext-1.3.jar. (2021, March 13). Www.java2s.com. http://www.java2s.com/Code/Jar/i/Downloaditext13jar.htm#google_vignette

Unique Developer. (2021, April 29). *AdminPanel with graphs - Google Drive*. Drive.google.com. https://drive.google.com/drive/folders/1Jmj4MH42GhmQeWVJ8HXIB7QeTpzzGRwI

Unique Developer. (2021, April 29). *JAVA - How to design a simple dashboard UI using Swing and Java with graphs -Netbeans 2021*. Www.youtube.com. https://www.youtube.com/watch?v=HviIu-9i1BU

Gupta, L. (2022, January 25). *Java - Read and Write PDF with iText*. HowToDoInJava. https://howtodoinjava.com/java/library/read-generate-pdf-java-itext/