MCIT 591
Final Project Report
Date: April 20, 2020

Project Title: **Tesla: Saving Hard-earned Cash or Not?**
Team Members: **Dewei Zhou** zdw466@seas.upenn.edu
Kelly Jackson Charles kcharl@seas.upenn.edu
Bingqian Lu bingqian@seas.upenn.edu

Introduction

Electric vehicles (EVs), such as the car brand Tesla, use kWh (the measure of energy from electricity) to measure energy use. The EV's *Miles Per Gallon equivalent* (MPGe) represents the number of miles the vehicle can travel using the same energy content as gasoline. This metric also signifies the maximum distance an EV can travel on a full charge. One way to compare energy costs between the EV and an internal combustion engine (ICE) vehicle is to convert the amount of energy in a gallon of gasoline into kWh of energy. Based on a formula from the US Environmental Protection Agency (EPA), 33.7 kWh of electricity is equivalent to one gallon of gasoline, rounded up to 34kWh. Studies show that a gasoline vehicle can use as much as 2.5 times more kWh of energy to travel the same distance as an EV. And, as gasoline and electricity cost change over time, so will the expense of operating each kind of vehicle. Because of these fluctuations in gasoline and electricity cost, drivers of EV and ICE cars may need easy-to-retrieve cost comparisons for specific vehicle models in order to determine the most cost-effective car purchase in their city.

Purpose

Therefore, this team's project has the intended outcome of using internal combustion engine (ICE) vehicle information on the Mercedes GLC and Lexus LX vehicles, as well as the current national average gas price per gallon, the average gas-powered car cost per mile to run, gasoline grade costs, and stated mpg to calculate whether driving a Tesla Model 3 is a cost-savings strategy in select east coast cities, and specifically in Miami FL, Charlotte NC, and New York NY.

Description of the Programming Language

The programming language used in this script is Java, an object-oriented programming language. Additionally, the team utilized Java SE 12 Development Kit (JDK) and the Eclipse Integrated development environment to manage, compile, load, verify and execute the program. This Java program is designed to display output messages to the console, in a text file, and in chart form. Additionally, the program obtains input from users, calculates formulae and saves their results for use later. Lastly, the program compares numbers, then displays messages that show the comparison results.

Market Use

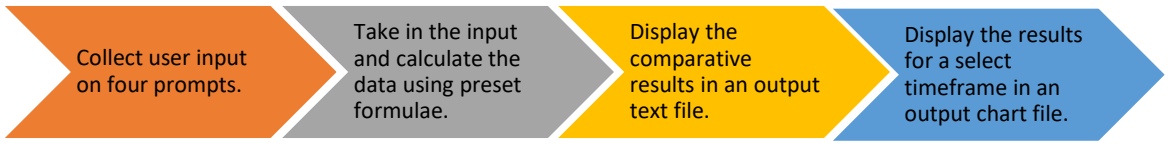
This Tesla program is easy to install and use, and will be marketable to consumers who are seeking to compare luxury vehicle energy costs, and specifically the costs of driving Tesla, an electric vehicle and two ICE vehicles, Lexus LX and Mercedes GLC. Consumers who may be most interested in this program include those New York NY, Charlotte NC, and Miami FL drivers

interested in the energy costs of the Tesla Model 3 compared to driving a Lexus LX or Mercedes GLC utilizing regular and premium gasoline. This is a free version of the program, and it promises user engagement with personalized metrics.

Methodology

The methodology for this team Java project includes the following activities: collecting user input, displaying results in the console, a text file and a chart, calculating formulae, and storing values for later use. See *Chart 1: Logic of the Project* below. This project was built to include three Excel csv files which consist of two electricity bills from one Tesla owner, available data for one Mercedes and Lexus owner, and Miles Per Gallon and gasoline cost details from the US Department of Energy. The csv files are titled: *09-2018--03-2019, 09-2019--03-2020* and *Gas_Data_EastCoastOnly*.

Chart 1: Logic of the Project

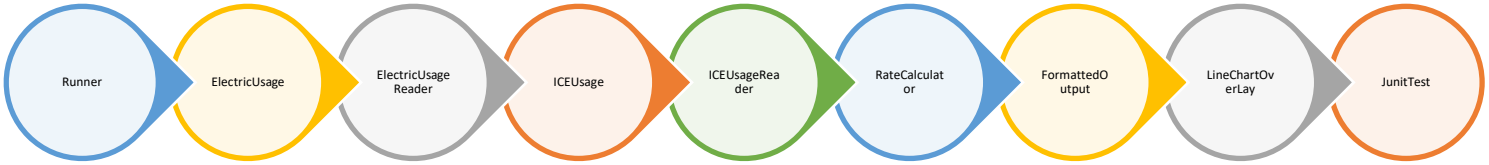


There are five main data structures comprising this project:

1. Insert – algorithms to insert user input into formulae,
2. Search – algorithms to search for values in csv files,
3. Store-algorithms to store values in arraylists and hashmaps in order to pair and organize data,
4. Output-an algorithm to output results to a text file, and
5. Graph-an algorithm to output chart display for user that incorporates predetermined values.

The user interface includes search fields in the console to retrieve customized input. The program also outputs a chart file with user details. There are several classes that organize the functions within the project, as shown in *Image 1: Order of Classes* below.

Image 1: Order of Classes



Class Functions and Rationale

Details on each class are offered below.

1. ElectricUsage: designed to list get methods for electricity data in order to encapsulate getters
2. ElectricUsageReader: designed to retrieve electricity data from csv files in order to read and store data streams. See the *Sample of ElectricUsageReader Class* below.

ElectricUsageReader - Notepad

File Edit Format View Help

```
import java.util.*;
import java.io.*;

/**
 * This class will read electricity usage csv file and store each column in each
 * arraylist.
 */

public class ElectricUsageReader {

	/**
	 * Initialize arraylists as instance variables, which can be used in other classes.
	 * listmonth = arraylist contains the months for each day.
	 * monthofdata = arraylist contains only the unique months, in our project will be
	 * (Sep,Oct,Nov,Dec,Jan,Feb).
	 * dayofdata = arraylist contains the date of each month for 6 months.
	 * usageofdata = arraylist contains the daily usage in kwh for 6 months.
	 * costofdata = arraylist contains the daily usage in dollar for 6 months.
	 * monthlyusage = arraylist contains the monthly usage in kwh for 6 months.
	 * monthlycost = arraylist contains the monthly usage in dollar for 6 months.
	 */

	ArrayList<Integer> listmonth = new ArrayList<Integer>();
	ArrayList<Integer> monthofdata = new ArrayList<Integer>();
	ArrayList<Integer> dayofdata = new ArrayList<Integer>();
	ArrayList<Integer> usageofdata = new ArrayList<Integer>();
	ArrayList<Double> costofdata = new ArrayList<Double>();
	ArrayList<Integer> monthlyusage = new ArrayList<Integer>();
	ArrayList<Double> monthlycost = new ArrayList<Double>();
	/**
	 * Initialize arraylists for monthly cost in $ from Sep to Feb
	 */
	ArrayList<Double> sepCost = new ArrayList<Double>();
	ArrayList<Double> octCost = new ArrayList<Double>();
	ArrayList<Double> novCost = new ArrayList<Double>();
	ArrayList<Double> decCost = new ArrayList<Double>();
	ArrayList<Double> janCost = new ArrayList<Double>();
	ArrayList<Double> febCost = new ArrayList<Double>();
	/**
	 * create one HashMap to store data through csv files, and convert data to int
	 * or double.
	 */
```

Sample of ElectricUsageReader Class

3. FormattedOutput: designed to format output into text file in order to print results to a save-able document
4. ICEUsage: designed to list get methods for ICE data in order to encapsulate getters
5. ICEUsageReader: designed to retrieve ICE data from csv file in order to read and store data streams
6. JunitTest: designed to test 15+ methods across at least 3 classes in order to ensure code compiles. See the *Sample of Junit Class* below.

JunitTest - Notepad

File Edit Format View Help

```
import static org.junit.jupiter.api.Assertions.*;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.junit.jupiter.api.Test;

class JunitTest {

	@Test
	void TestMonthlyUsage() {
		ElectricUsageReader TeslaUsage = new ElectricUsageReader("09-2019--03-2020.csv");
		ArrayList<Integer> monthlyusage = new ArrayList<Integer>();
		monthlyusage.add(1027);
		monthlyusage.add(694);
		monthlyusage.add(706);
		monthlyusage.add(1123);
		monthlyusage.add(689);
		monthlyusage.add(734);
		TeslaUsage.list();
		TeslaUsage.MonthlyUsage();
		assertEquals(monthlyusage, TeslaUsage.getMonthlyusage());
	}

	@Test
	void TestMonthlyCost() {
		ElectricUsageReader TeslaUsage = new ElectricUsageReader("09-2019--03-2020.csv");
		ArrayList<Double> monthlycost = new ArrayList<Double>();
		monthlycost.add(94.63);
		monthlycost.add(63.92);
		monthlycost.add(65.38);
		monthlycost.add(103.55);
		monthlycost.add(63.44);
		monthlycost.add(67.67);
		TeslaUsage.list();
		TeslaUsage.MonthlyCost();
		assertEquals(monthlycost, TeslaUsage.getMonthlycost());
	}

	@Test
	void TestEachMonthlyCostFeb() {
```

Sample of Junit Class

7. LineChartOverLay: designed to output chart display in order to increase user understanding of data
8. RateCalculator: designed to calculate values using formulae in order to systematically perform the basic mathematical operations of the program
9. Runner.java: designed to start the program in order to execute the program's primary functions. See the *Sample of Runner Class* below.

Runner - Notepad

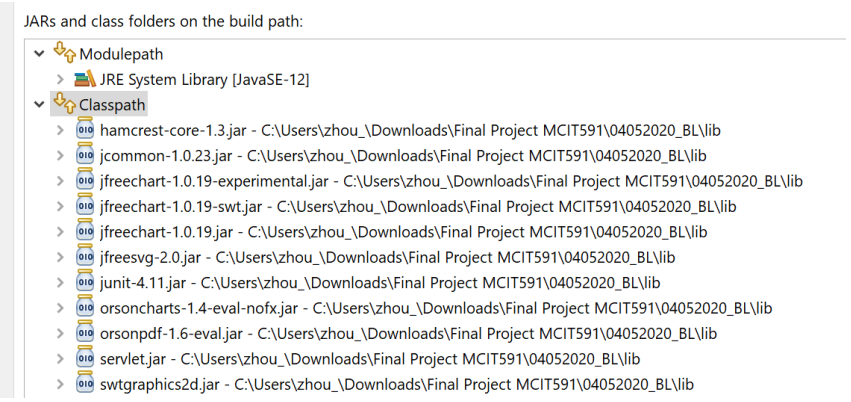
File Edit Format View Help

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.category.DefaultCategoryDataset;

public class Runner {

	public static void main(String[] args) {
		/**
		 * input file names and create new Object.
		 */
		ElectricUsageReader usageWithoutEV = new ElectricUsageReader("09-2018--03-2019.csv");
		ElectricUsageReader usageWithEV = new ElectricUsageReader("09-2019--03-2020.csv");
		RateCalculator vehicleSpecification = new RateCalculator();

		/**
		 * ask user for input the 3 parameters: mile, year and % check whether or not
		 * the input is invalid.
		 */
		double mileage = 0.0;
		double totalYear = 0.0;
		double percentage = 0.0;

		Scanner in = new Scanner(System.in);
		// prompt user to input mileage info ------------------------------------
		System.out.println("Please enter the mileages you normal drive in one year:");
		mileage = in.nextDouble();
		while (mileage <= 0) { // check if mileage input is valid
			System.out.println("Invalid input: Mileage should be above 0. "
					+ "Please try again: enter the mileages you normal drive in one year");
		mileage = in.nextDouble();
		}
		// prompt user to input totalYear info-----------------------------------
		System.out.println("Please enter the total years you would like to drive your car:");
		totalYear = in.nextDouble();
```

Sample of Runner Class

To start the project, the user begins in the *Runner Class* and clicks on the main method to initiate the program. Eclipse Version 12 or higher is required to execute the code, and there are eleven libraries that must accompany the script, as shown below in *Image 2: Library Files*.

Image 2: Library Files



Potential Design Improvements:
The team understands that the goal of writing Java code is to create a program that is readable and DRY. If given more time to improve this program, we would work to:

1. Include both east and west coast cities in our analysis of ICE and EV comparisons,
2. Include other vehicle models and manufacturers in our data structures, such as Toyota,
3. Improve the variable names in each class to better reflect their use in the program,
4. Be more consistent with indentation throughout the program, and
5. Match close braces with the statements being closed throughout the program.

Conclusion:
In conclusion, the "Tesla: Saving Hard-earned Cash or Not?" program's methodology is based on the user's active participation and interaction, and offers multiple outputs for the development of a cost comparison between the electric vehicle Tesla Model 3 and ICE vehicle information on the Mercedes GLC and Lexus LX vehicles, as well as the current national average gas price per gallon, the average gas-powered car cost per mile to run, gasoline grade costs, and stated mpg in select east coast cities, and specifically in Miami FL, Charlotte NC, and New York NY. Using Java, three csv files, Eclipse and select libraries, our team's goal as developers was to help consumers who may be interested in analyzing the energy costs of the Tesla compared to driving a Lexus LX or Mercedes GLC automobile utilizing regular and premium gasoline. The project revealed that the Tesla vehicle is a consumer-savings option in terms of energy cost in comparison to the Mercedes GLC and Lexus LX in three select east coast cities. As an example, when the user entered that they would drive the Tesla for 12,000 miles over a 12-month period, and with 65% driving time on the highway, a significant annual savings emerged as shown below in Table 1.

Table 1: One-Year Costs by City

	Charlotte	New York	Miami
Tesla	$305.25	$652.43	$396.11
Mercedes	$1673.27	$2072.04	$2220.60
Lexus	$1378.67	$1762.50	$1699.83

Further, when the user entered that they would drive a Tesla for 12,000 miles a year, over 8-years, and with 65% driving time on the highway, a more significant savings for the driver was revealed as shown below in Table 2. Lastly, on average, Tesla will save consumers 84.81% in energy costs when compared to Mercedes GLC drivers, and 79.18% when compared to Lexus LX drivers.

Table 2: Eight-Year Costs by City

	Charlotte	New York	Miami
Tesla	$2441.97	$5219.40	$3168.89
Mercedes	$13386.14	$14100.00	$17764.78
Lexus	$11029.33	$16576.29	$13598.67

Overall, this free Java program with multiple classes and methods demonstrates DRY code that is innovative and useful for today's cost-conscious car buyer, and the program establishes a clear savings when select east coast drivers opt for the electric vehicle Tesla.

References:

US Environmental Protection Agency. New Fuel Economy and Environment Labels for a New Generation of Vehicles. Retrieved from epa.gov.

US Department of Energy. Alternative Fuels Data Center. Retrieved from: https://afdc.energy.gov/calc/cost_calculator_methodology.html