Cruise Control Software Development Version 0.11

We pledge our honor that we have abided by the Stevens Honor System.

CS347: Software Development Processes — Spring 2020

Team Mike

Eric Altenburg
ealtenbu@stevens.edu

Michael McCreesh
mmccree1@stevens.edu

Hamzah Nizami
hnizami1@stevens.edu

Constance Xu
cxu16@stevens.edu

Contents

1 Executive Summary

Team Mike is a startup initiative aimed at solving problems that come to light as society begins to adopt new technologies. One of which pertains to autonomous driving and "smart cars" as they are beginning to break into the automobile market more and more every year with examples such as Tesla and Ford. In a perfect world, if every driving car were to be a smart car with "autopilot," then it would make sense for each of them to communicate cruise control data with each other to reduce traffic build-up and make traveling more efficient. Through rotational leadership on a monthly basis, each team member possesses a set of distinct skills that can translate to highly efficient work sessions. Aside from developing a traditional cruise control, the aim is to make the software open source as it will serve as the foundation to a more interconnected logistical future in which autonomous cars can fully achieve their potential within a growing technologically advanced society.

2 Introduction

The past century has seen an explosion of innovation on a scale never before seen in human history. The rapid development and refinement of a myriad of motor technologies catalyzed the innovation process by allowing ideas to transfer at rates never before seen. With the constraint of distance loosening thanks to every iteration of the automobile, people have been granted more freedom to do what they please. While the automobile has enhanced society in several ways, there are some glaring problems that need to be addressed to continue the current rate of human progress and innovation. One of the biggest problems is driver fatigue, which is responsible for approximately 72,000 crashes annually. Programming the automobile to be reliably autonomous to a degree is one way to circumvent the issue of driver fatigue and making roads safer. That's exactly what cruise control aims to do. By moderating the speed of the vehicle by itself, cruise controls aim to lessen the effects of driver fatigue on the road. However, the technology is not perfect, and Team Mike aims to resolve that by creating the perfect cruise control system.

When it was implemented in 1958, cruise control was suitable for that era, however, as technology continually improves, the embedded software must as well. For longer car trips, it does not make sense for users to constantly accelerate and decelerate for several miles; this is where cruise control comes in. By being able to set the speed, the user is now able to accurately set and maintain the speed they wish to without having to constantly intervene which, over time, would cause less harm to the vehicle as variable speed and RPMs are not as desirable as that of near-constant.

Our group intends to use an Agile method to implement our version of cruise control. This would encourage code sprints in two-week lengths where work will be divided into smaller sections and distributed based on each members' strengths. The Agile method may vary as to which one we specifically choose but regardless, this project will not come to fruition using Waterfall or other methods. The way that we are going to organize our group is through weekly meetings and ensuring that everyone knows their tasks for the week. These weekly meetings can also serve as a place where team members mention any obstacles that have come in their way when trying to resolve a problem, and also serve as a time where the team as a collective can try to think of solutions around those obstacles. We also plan on giving real-time updates about any progress made through Slack in order to ensure all members of the team are equipped with the most recent developments of the project. We also intend on using Java to implement our cruise control

system. We believe that the high-performance Java provides and the fact that its part of the Object-Oriented Programming (OOP) paradigm makes it perfect for embedded systems.

There are a plethora of features that make up cruise control. As most cruise controls are seen on vehicles, there are generally a few buttons: increase speed (and sometimes decrease speed) and the button that starts cruise control. Once cruise control is set, the user then has the ability to increase, decrease, or maintain the speed of a vehicle. Furthermore, if the user presses on the brake, then cruise control is automatically deactivated. The system also allows for the operator of a vehicle to manually deactivate it.

To successfully implement the features of maintaining a steady speed and a safe distance away from other cars, there are certain requirements to fulfill. At a high level, proper hardware with fast, reliable sensors is critical for this project. Furthermore, lives are fundamentally dependent on this product and therefore it requires software that is error-resistant and thus a strong development and QA (quality assurance) team. In addition, the software must also perform it's expected tasks of moderating speed, maintaining a safe distance and switching back control to the driver when prompted to quickly. Having a cruise control software that is slow would be ineffective, so speed and accuracy are critical requirements for the software as well as the hardware. This is a mission-critical system because if it were to fail, it would put the lives of people in danger.

3 Requirements

3.1 Input

1. The system shall accept electric power from the alternator.
2. The power button that allows for the state of the system to change from on to off or vice versa.
3. When pressed, a button will either accelerate or decelerate in 1 mph increments.
4. When the brake is pressed, the cruise control system will unset the speed and give control back to the user until the user specifies a new speed to be set.
5. If the entire car turns off, then a signal will be sent to turn off the cruise control system.
6. RPM (Rotations Per Minute) sensor should be connected to the front axle and able to take readings for accurate speed calculations.
7. Engine sensor must be able to take input from the engine to tell the cruise control system to turn on or off.
   (a) If the engine is turned on, the cruise control system must be ready for use within 3 seconds of the engine being turned on.
8. If the gas pedal is pressed, the vehicle will continue to accelerate at the control of the user, but when the gas pedal is released the cruise control system will continue to the previously set speed.

3.2 Output

1. Keep a log of activity in the system files to help debug in the event of a malfunction.
2. For every speed increase made by the user in the cruise control system, a visual indication by the software is necessary. This is so that the user minimizes their own human error. So, every time you increase the speed, you can see the cruise control speed on the car menu increasing by however much the user wants it to.
3. The system displays successful activation or deactivation in 10 milliseconds.

(a) Numbers are subject to change depending on how inputs are received from surrounding system, but are expected to be in around that same ballpark.

4. Keep a log of every time the cruise control system is used and at what speed written to the system files.

5. Once the cruise control system is turned on, it must be readily available. The engine sensor must be able to understand that the engine is on and tell the cruise control system that, if the user so wishes, it must activate.

6. With all the inputs it is taking from all the sensors, the software must be able to deliver the desired output for each function in less than 15 milliseconds.

   (a) The brake pedal sensor must be able to tell the software that the user has stopped, and stop the cruise control system. *Numbers are subject to change depending on how inputs are received from surrounding system, but are expected to be in around that same ballpark.*

3.3 Functional

1. The system shall accept direct current from the car battery to support logging after engine shut down.

2. The system shall receive the time and date from the car's clock every second.

3. A physical will be provided by the system for technicians to access the unit.

4. Hardware shall have a 4 nine (99.99%) availability.

5. Software shall have a 5 nine (99.999%) availability.

6. While the cruise control system is on and a speed is set, it must be able to increase and decrease the speed by 1 mph.

7. Able to switch the cruise control system on or off provided the engine is on.

8. Interpret engine on and off signal to allow for the cruise control system to be turned on within 3 seconds of received the signal.

9. Only allow the cruise control system to be activated when at a minimum speed of 25 mph.

    (a) If the speed is set to 25 mph, the user cannot decrease the speed below 25 mph.

10. Maximum speed that cruise control system can be set to is at 125 mph.

11. While the cruise control system is on, if a user is driving at a speed of at least 25 mph and decides to set the speed, the cruise control system will maintain that speed.

12. If the user presses and continues to press the brakes, the user will not be able to set the speed.

13. If the user presses and continues to press the gas pedal, the user can set the current speed for the cruise control system, but it will immediately pause as stated in the input requirements then resume after the user releases the gas pedal.

3.4 Security

1. No external interface to reduce potential tampering. This applies to both the hardware such as sensors and the general cruise control system software. There would be no easy access to the cruise control system hardware or software so that the likelihood of someone being able to create issues is reduced substantially.

2. No Internet or Bluetooth connection. This is so that no bad actors can meddle with the car's system and hence, keeps the drivers safer as cyber security becomes more of an issue.

3. An administrator will need hardware to make changes to the mechanisms. The administrator must be authorized to make these changes and they must be a trusted third party or those who created the cruise control system themselves.

4 Requirements Analysis Model

4.1 UML Use Cases & Diagram

See Figure 1 for general use cases of cruise control system.

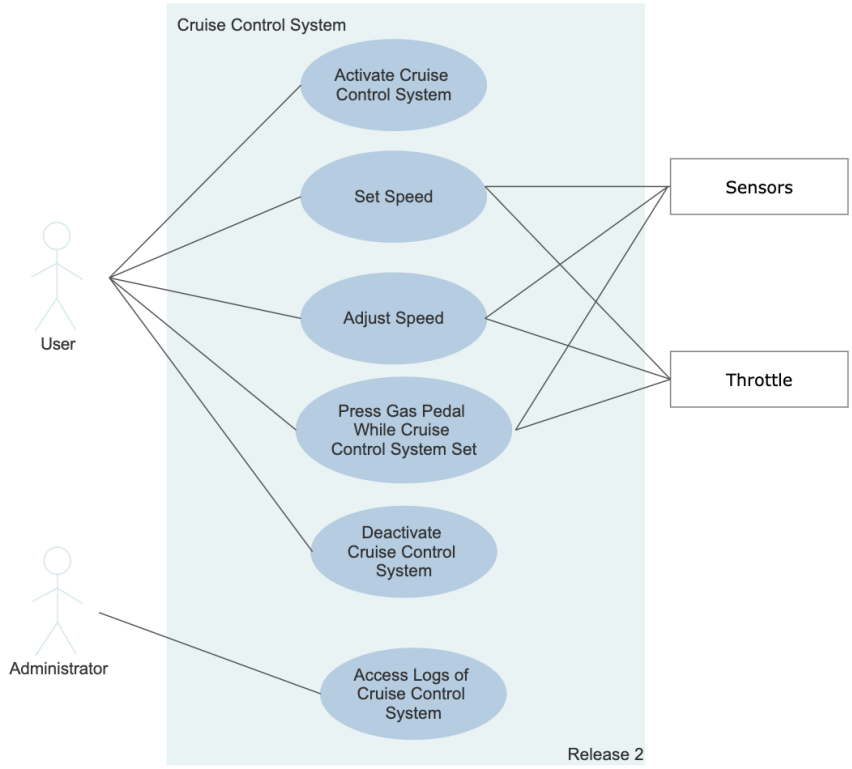

Figure 1: Sample UML diagram for cruise control system use cases.

It should be mentioned that there are two similar sounding states that need to be clarified:

- Activated
  - Granted the vehicle is on, the cruise control system is also now waiting for the speed to be set by the user.
- Set/Unset

- Given that the cruise control system has been **activated**, the user has requested for the speed to be maintained at its current speed.
- Given that the cruise control system has been **activated** and the cruise control system is in the **set** state, the user can either press the brake or press a button to unset the speed and revert back to the **activated** state.

1. Use Case 1:

   Primary Actor: User

   Description: User activates the cruise control system

   Outcome: The cruise control system is now activated

   Basic Flow:

   (a) Given that the vehicle is on, the user requests an activation of the cruise control system.

   (b) Cruise control system activates.

   (c) The cruise control system provides a visual feedback that it has been activated.

2. Use Case 2:

   Primary Actor: User

   Description: User sets the speed

   Outcome: The speed of the cruise control System is set to a specific speed

   Basic Flow:

   (a) Given that the vehicle is on and the cruise control system is in the activated state, the system requests values from sensors.

   (b) Sensors provide values for approval to set cruise control system at current speed.

   (c) Cruise control system requests the Engine Management System (EMS) set the speed at current position.

   (d) EMS Speed (Throttle) is set at current speed.

   (e) Cruise control system provides visual feedback to the user that the cruise control is set and working.

(f) Sensors provide the changing environmental information to the cruise control unit (such as speed, request for increase/decrease speed, and brake).

(g) Cruise control system detects the changes from the sensors and request adjusting speed or deactivating Cruise Control system accordingly.

(h) Speed (Throttle) position is continuously set to new values to ensure that the speed remains constant.

(i) Speed is continuously reported to the cruise control system.

3. Use Case 3:

   Primary Actor: User

   Description: User increases speed

   Outcome: The speed of the Cruise Control System is increased

   Basic Flow:

   (a) If the cruise control system is in the set state, the user requests an increase of cruise control system speed.

   (b) Cruise control system provides visual feedback that the system will alter the speed of the vehicle.

   (c) Cruise control system slowly increases the speed of the vehicle to match that of the request.

   (d) When the desired speed is reached, the cruise control system will provide visual feedback that the adjustment has been completed.

4. Use Case 4:

   Primary Actor: User

   Description: User decreases speed

   Outcome: The speed of the Cruise Control System is decreased

   Basic Flow:

   (a) If the cruise control system is in the set state, the user requests a decrease of cruise control system speed.

   (b) Cruise control system provides visual feedback that the system will alter the speed of the vehicle.

   (c) Cruise control system slowly decreases the speed of the vehicle to match that of the request.

(d) When the desired speed is reached, the cruise control system will provide visual feedback that the adjustment has been completed.

5. Use Case 5:

   Primary Actor: User

   Description: User presses gas pedal while the cruise control system is set

   Outcome: The speed of the vehicle increases while the gas remains pressed. When the gas is released, the speed will return to the speed of the Cruise Control System

   Basic Flow:

   (a) If the cruise control system is in the set state, the user presses on the gas pedal.
   (b) In the event that the vehicle accelerates, the cruise control system will continually request the EMS to be set to the previously specified speed.
   (c) Speed will be continuously reported to the cruise control system.
   (d) After the user releases the gas pedal and the vehicle stops accelerating, the cruise control system will request for the EMS to be set to the previous specified speed.
   (e) The vehicle will naturally decelerate nearing the old set speed, and once reached, the cruise control system will resume with the set speed.

6. Use Case 6:

   Primary Actor: User

   Description: User applies brake while the cruise control system is set

   Outcome: The speed of the vehicle decreases while the brake remains applied. When the brake is released, the speed will return to the speed of the Cruise Control System

   Basic Flow:

   (a) If the cruise control system is in the set state, the user presses on the brake.
   (b) In the event that the vehicle decelerates, the cruise control system will continually request the EMS to be set to the previously specified speed.

(c) Speed will be continuously reported to the cruise control system.

(d) After the user releases the brake and the vehicle stops decelerating, the cruise control system will request for the EMS to be set to the previous specified speed.

(e) The vehicle will naturally accelerate nearing the old set speed, and once reached, the cruise control system will resume with the set speed.

7. Use Case 7:

   Primary Actor: User

   Description: User deactivates cruise control system

   Outcome: The Cruise Control System is now deactivated

   Basic Flow:

   (a) If the cruise control system is activated, the user requests a deactivation of the cruise control system.

   (b) Cruise control system deactivates.

   (c) Cruise control system provides visual feedback that it has been deactivated.

8. Use Case 8:

   Primary Actor: Administrator

   Description: Administrator accesses logs of cruise control system

   Outcome: The administrator is given access to the log of the Cruise Control System

   Basic Flow:

   (a) With the vehicle turned on and the cruise control activated, an administrator will have to use a proprietary physical hardware key in order to gain root access.

   (b) Once access has been granted, the administrator can download the logs to an external storage device through a USB port.

   (c) Once downloaded, the administrator must log out using the same proprietary physical hardware key.

4.2 UML Class-Based Modeling

1. See Figure 2 for the class-based modeling for the cruise control system.

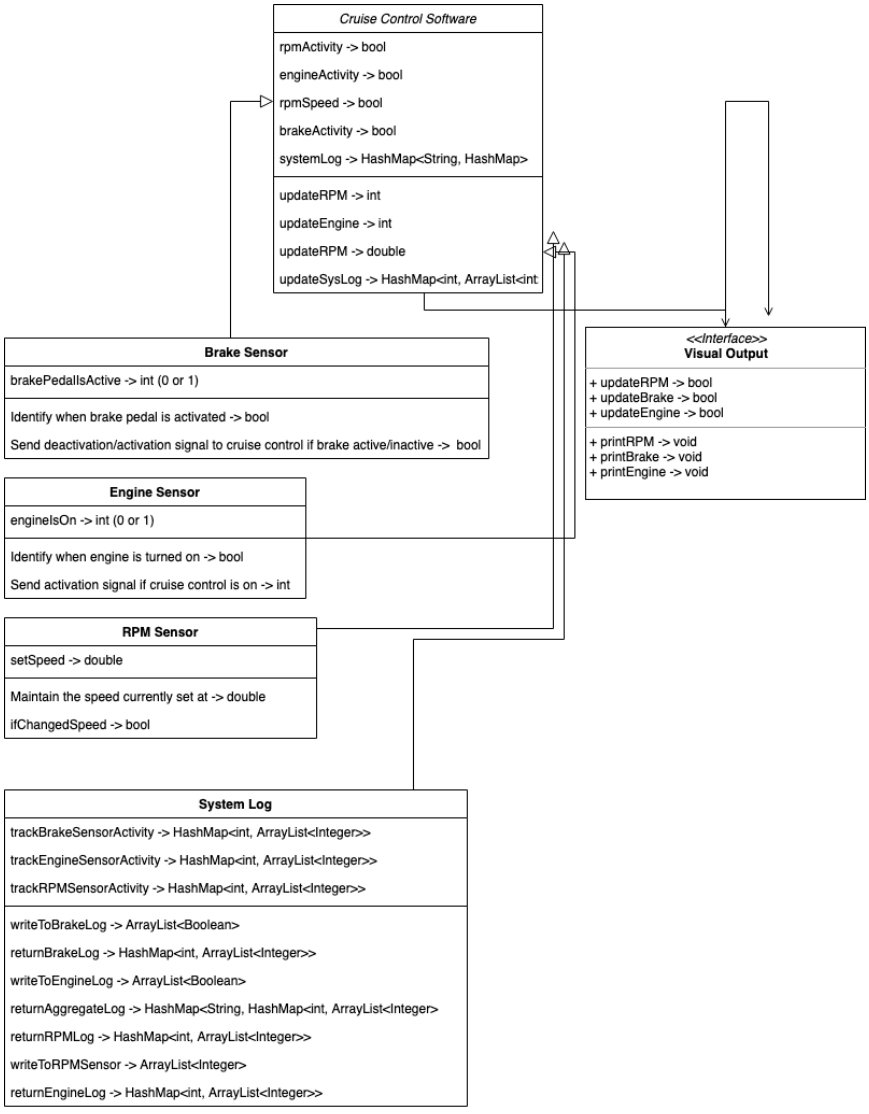

Figure 2: Sample UML class-based model for the cruise control system.

4.3 UML CRC Model Index Card

1. See Figure 3 for the CRC Model Index Cards for the cruise control system.

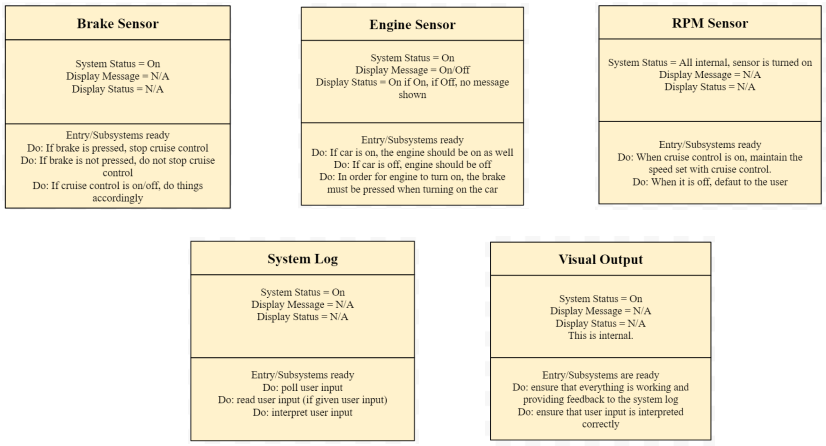

Figure 3: Sample UML CRC model index cards for the cruise control system.

4.4 UML Activity Diagram

1. See Figure 4 for the activity diagram of the cruise control system.

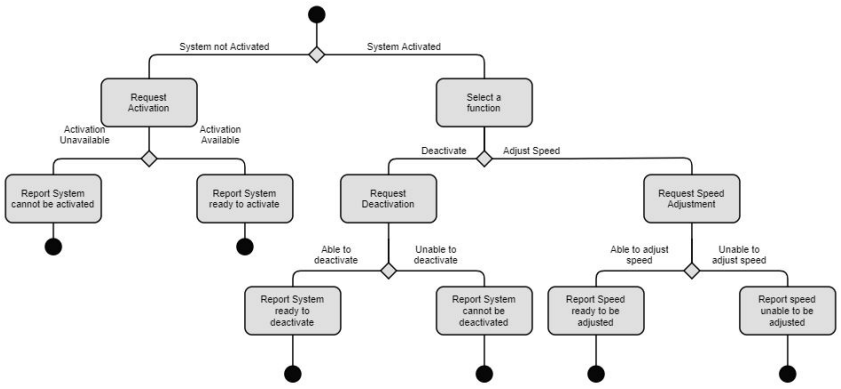

Figure 4: Sample UML activity diagram for the cruise control system.

4.5 UML Sequence Diagram

1. See Figure 5 for sequence diagram of the activation of the cruise control system.

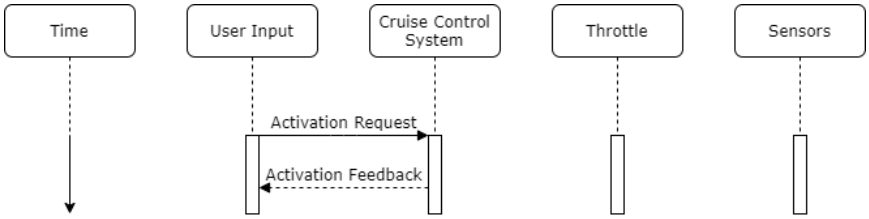

Figure 5: Sequence UML digram for activation of the cruise control system.

2. See Figure 6 for sequence diagram of cruise control system setting speed.

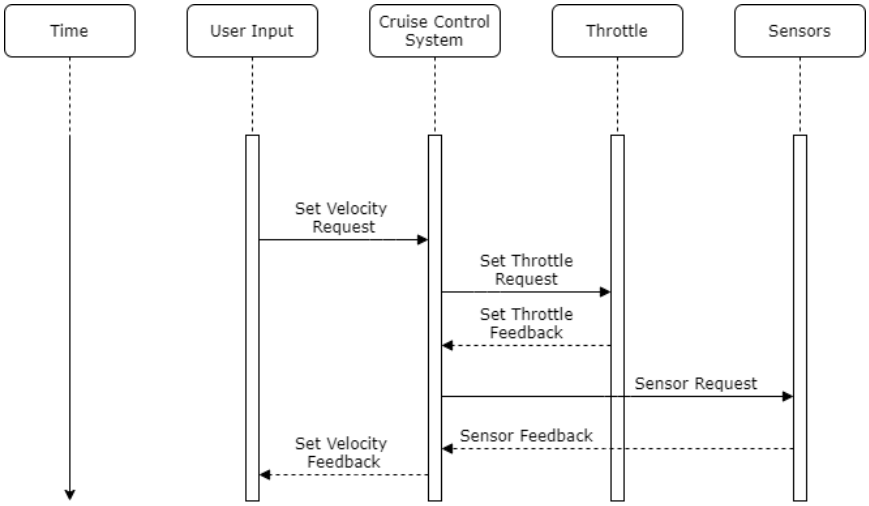

Figure 6: Sequence UML diagram for the cruise control system setting speed.

3. See Figure 7 for sequence diagram of cruise control system adjusting speed.

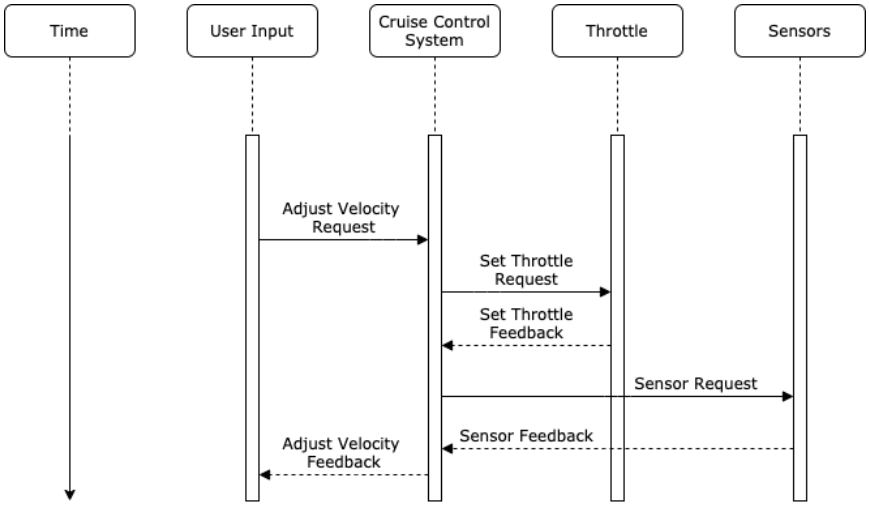

Figure 7: Sequence UML diagram for the cruise control system adjusting speed.

4. See Figure 8 for sequence diagram of the cruise control system being suspended when the user presses the gas pedal.

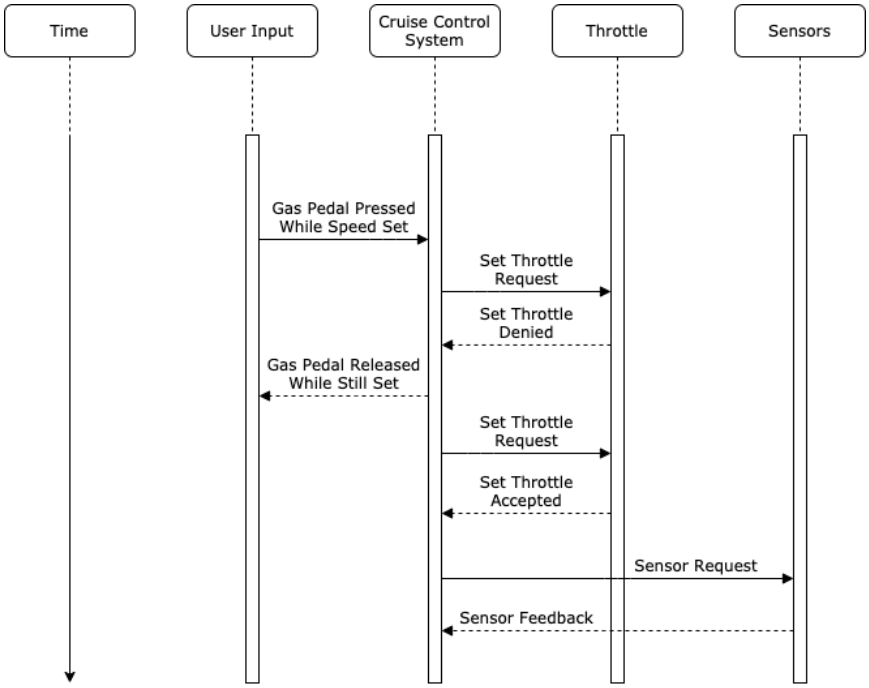

Figure 8: Sequence UML diagram for the cruise control system suspending during gas pedal being pressed.

5. See Figure 9 for sequence diagram of the cruise control system being deactivated.

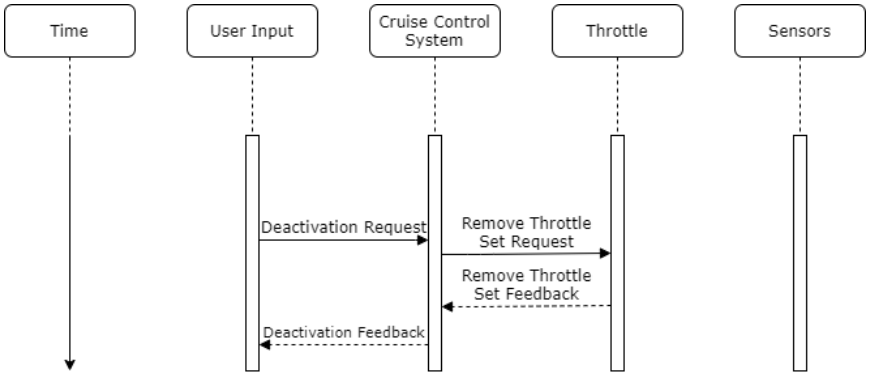

Figure 9: Sequence UML diagram for the cruise control system being deactivated.

6. See Figure 10 for sequence diagram of the administrator accessing the cruise control system logs.

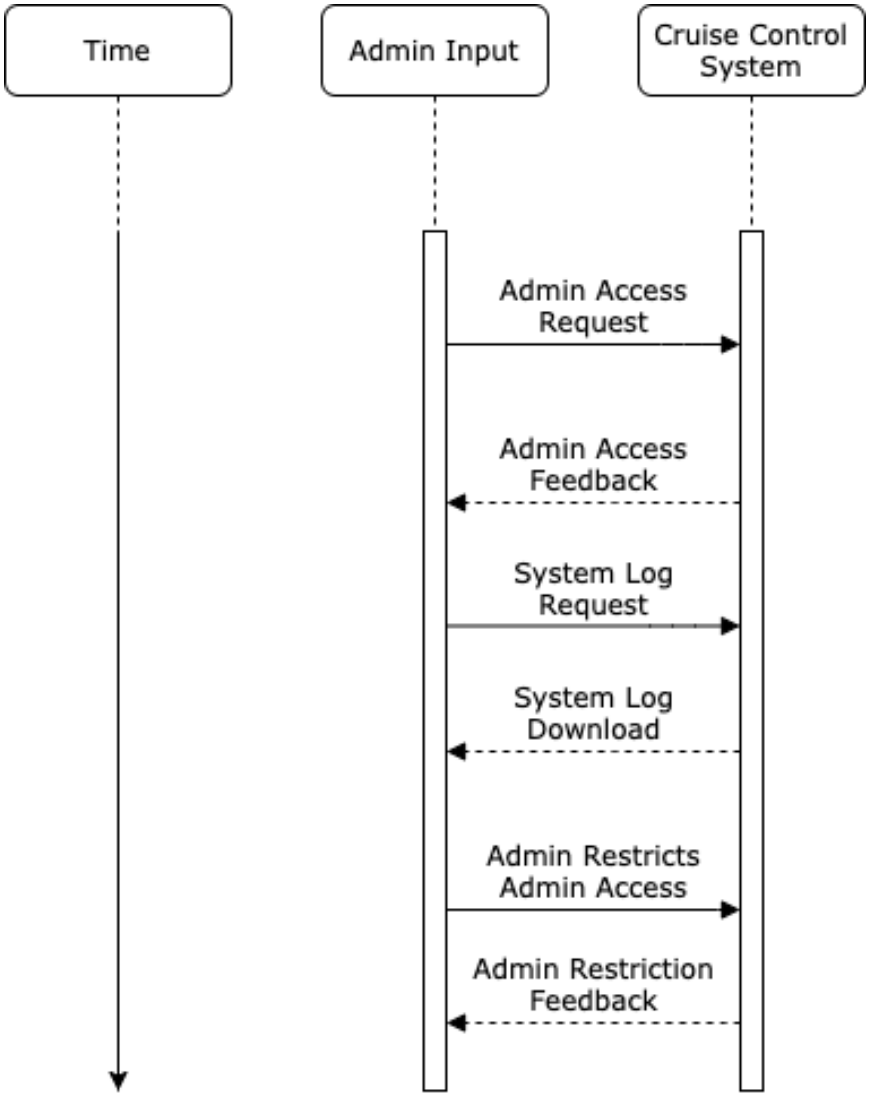

Figure 10: Sequence UML diagram for the administrator accessing cruise control system logs.

4.6 UML State Diagram

1. See Figure 11 for the state diagram of the cruise control system.

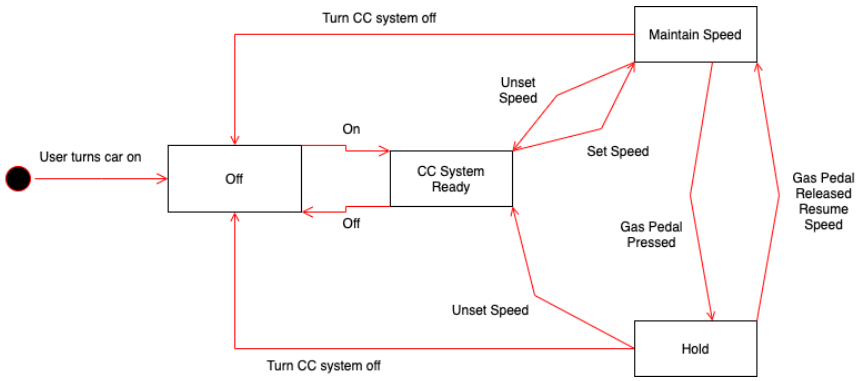

Figure 11: Sample UML state diagram for the cruise control system.

5 Software Architecture

5.1 Architecture Style

We decided to implement our Cruise Control System by using Object Oriented Programming.

1. We chose this architecture style because it will allow us to represent the different aspects of the cruise control system as different objects. For example, we will be able to represent the engine, the brake, the throttle, the sensors, and the log as different objects. These objects will have different attributes which will allow the program to closely resemble the real world. The brake object will contain a boolean that determines if the brake is being pressed or not. It will also have a function that will decrease the current speed of the vehicle. The sensors will have variables that determine if the current speed is less than, greater than, or equal to the desired cruising speed. The throttle will contain a boolean that determines if the user is trying to manually increase the speed of the vehicle. It will also have a function that will increase the speed of the car. The engine object will contain a boolean that determines if the vehicle is on or off. The log will be a tuple of a time stamp and a string. Whenever any action is taken by the driver or the cruise control system, it will be added to the log. All of these objects will be used as components of a larger Cruise Control System object.

2. Using object-oriented design will limit the number of programming languages that we would be able to use. Using object-oriented programming can result in more resources being used because the programs can become very large. It is also difficult to determine all of the classes that will be necessary in order to properly implement the Cruise Control System at the beginning of the project.

3. We also considered implementing our Cruise Control System with call and return architecture. This architecture design would have allowed us to split the main program into subprograms. It would have been easier to build upon the program by adding more subprograms. It would have also been able to set a hierarchy for different programs if we used the main program/subprogram style of this architecture.

4. We opted not to use a call and return architecture because this architectural design often does not adequately support the data structures

which would be important for the implementation of the Cruise Control System. Furthermore, due to a large number of subprograms, it may have become difficult to navigate and manage.

5. We did not consider the following architectures:

   (a) The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components are accomplished via message passing.

   (b) Pros: It models the real world very well. With OOP, programs are easy to understand and maintain. OOP offers code re-usability. Already created classes can be reused without having to write them again.OOP facilitates the quick development of programs where parallel development of classes is possible.With OOP, programs are easier to test, manage and debug.

   (c) Cons: With OOP, classes sometimes tend to be over-generalized. The relations among classes become superficial at times. The OOP design is tricky and requires appropriate knowledge. Also, one needs to do proper planning and design for OOP programming. To program with OOP, the programmer needs proper skills such as that of design, programming and thinking in terms of objects and classes etc.

5.2 Components

A software component is an architectural entity that encapsulates a subset of the system's functionality and/or data. A set of components will perform a function that is required by a system. This restricts access to that subset via an explicitly defined interface. The components of a software system are the following (but not limited to):

1. Network and Internet services
2. Hardware level of operating system
3. Logical level of operating system
4. Graphics engine
5. User interface
6. System services

7. Command shell

8. System utilities

Regarding cruise control specifically, the main component of our cruise control is the engine management system. In order to do this, the cruise control will receive data from the sensor components in which it relays data from the components of the car (take for example, the brake and the sensors).

Components and connectors are used to accomplish a system's goal. This is expressed through an architectural configuration. More precisely, an architectural configuration is an association between components and connectors of software architecture. A connector is not equivalent to a component as components provide application-specific functionality while connectors provide application-independent interaction mechanisms.

Connectors are between the sensors to deliver the data to the cruise control module. This processes the data to determine the speed of the car. The connectors between the cruise control and the engine management system will continually deliver this data to each other.

A software constraint defines how components can be integrated to form the system. A software constraint is a restriction on the degree of freedom you have in providing a solution. Constraints are effectively global requirements, such as limited development resources or a decision by senior management that restricts the way you develop a system.

The information that is shared between the sensor components to the signal that the engine has been started must be delivered very quickly before cruise control can be turned on. The activation and deactivation of cruise control must be done in a specific time constraint. These are all constraints of the system.

5.3 Control Management

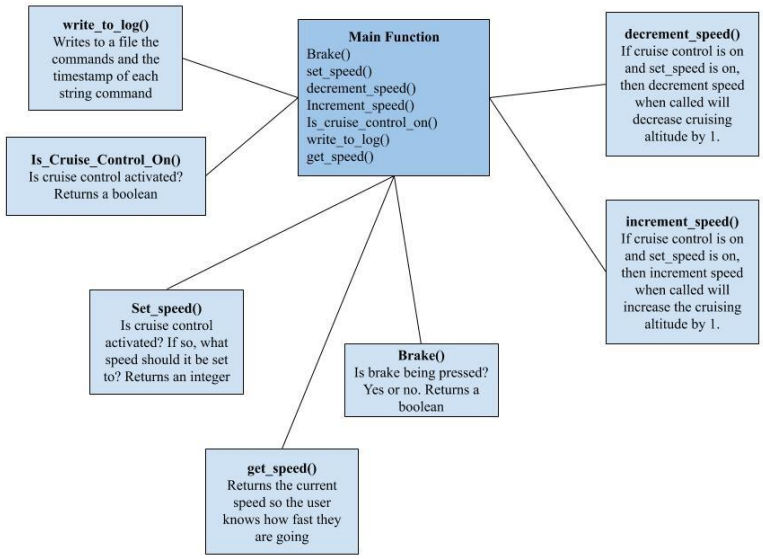

Figure 12: Control management diagram

We are going to have a main function that will have calls to the functions listed above as you can see: write_to_log(), Is_Cruise_Control_On(), Set_speed(), get_speed(), Brake(), increment_speed(), and decrement_speed(). These functions are essentially what are going to control our cruise control module. Brake will be a boolean as it is either true that the brake is being pressed or false if it is not being pressed. If cruise control is on and there is a set speed that is greater than 25 (which is the only way that cruise control can be activated in the first place), then if the brake function goes from false to true, then we shut off the cruise control entirely.

This is similarly listed out in the diagram as shown. All of our functions interact with our main function which is essentially the driver of our cruise control program.

5.4 Data Architecture

In terms of data architecture, considering that the project is going to be programmed in an object-oriented language such as Java, having the

data schema represented in the same programming paradigm streamlines the technical thought process behind the project. Having a data schema known for reliability is imperative for a mission-critical piece of software like cruise control.

Due to the use of Java, the general paradigm of the data architecture is object-oriented. Communication between the components is done via passing information as parameters to functional components such as methods and classes. Cruise control software requires information to be readily available to make critical decisions. An architecture based on passing information as parameters allows us to continually pass around data within the system, making data access trivially fast and facilitates quick decisions and computations that can mitigate the risk of fatal injury or things spiraling out of control.

To optimally implement this structure, we would need to define several functional components that each handles its own individual task and can be called as a function call to perform its individual task quickly. So, the cruise control system can have its own functional component for controlling the speed, controlling the brake, and sending visual and audio information to the user to inform about cruise control status such as whether it is on or off. The data components would always be available for use for functional components (such as program methods) in the cruise control software. This is because the functional components will need to repeatedly query the data components to accurately compute what speed is necessary, whether a brake is needed, and whether the user has requested for the system to turn on or off.

5.5 Architectural Designs

1. See Figure 13 for the architecture context diagram in which the external entities that the software interacts with are defined. Some include other systems, devices, and actors.

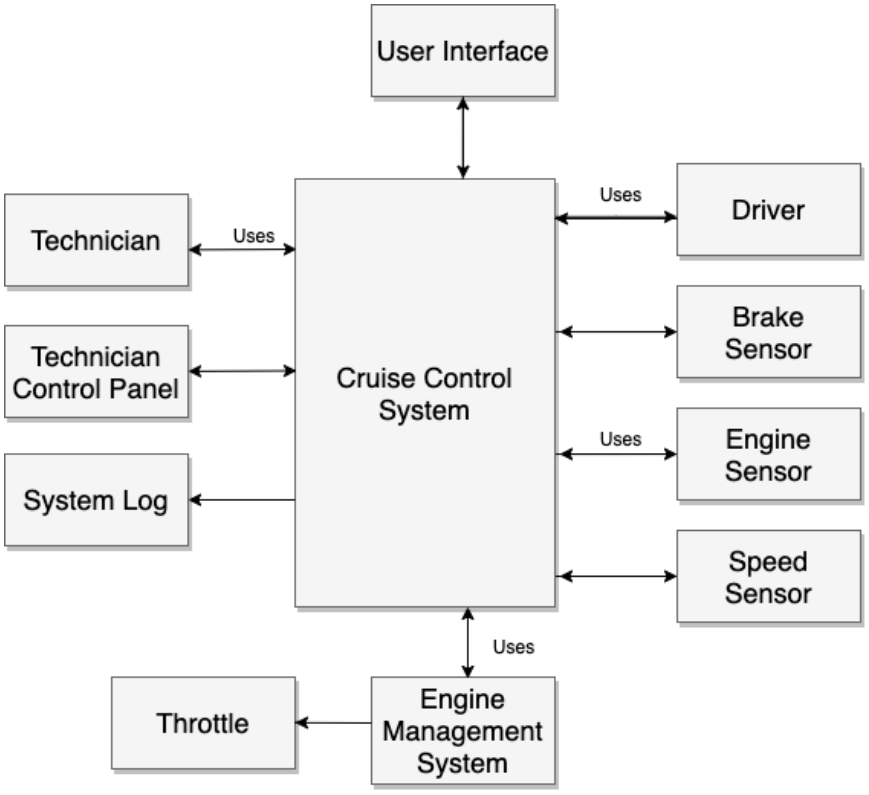

Figure 13: context diagram

2. See Figure 14 for the cruise control archetype. The archetype is an abstraction that represent an element of the cruise control system behavior.

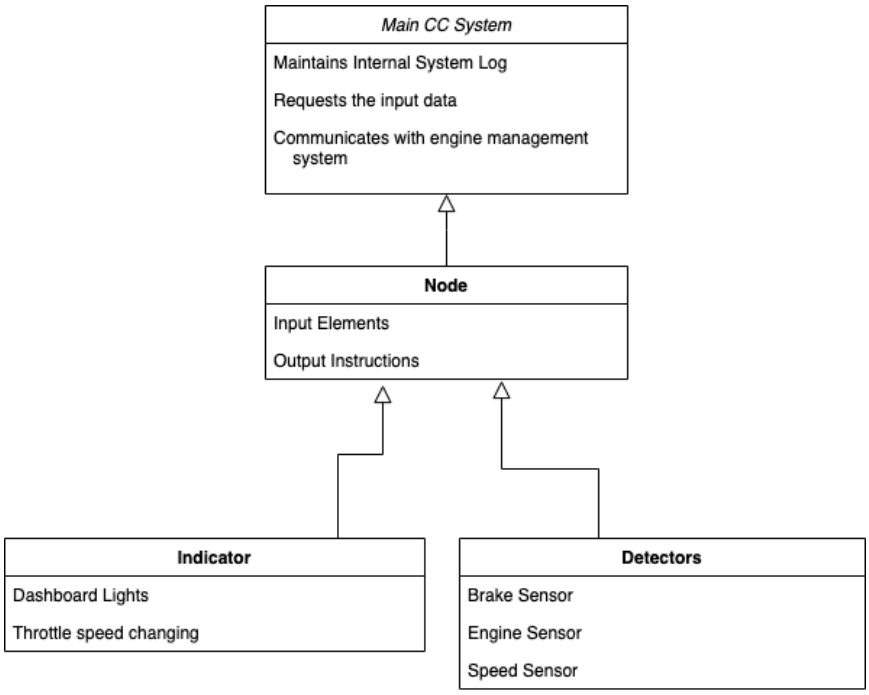

Figure 14: Cruise Control System Archetypes

3. See Figure 15 for the cruise control top-Level component in which the archetype is further defined for help with implementation.

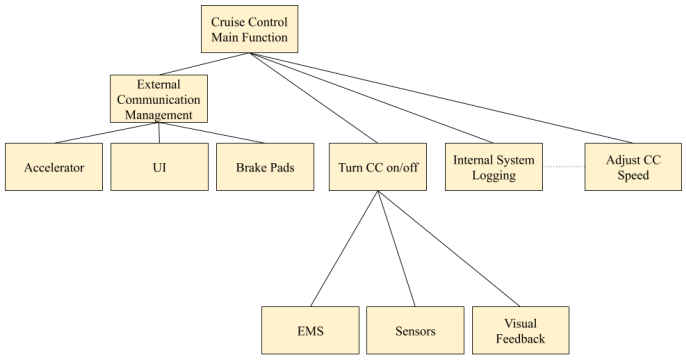

Figure 15: Top-level component

4. See Figure 16 for the refined component architecture. This is a more refined version of the top-level component to help with implementation as well.

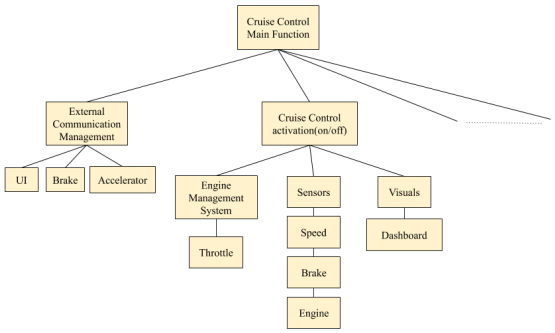

Figure 16: Refined component

5.6 Issues

1. Hamzah: The data architecture has not been thoroughly defined. Work through the data architecture and define how exactly the cruise control software will store the data using MongoDB in an object-oriented + call and return architecture. Review section 5.3.**Resolved: 4/17/2020**

2. Connie: work with Hamzah to identify how control interacts with the Data Architecture and the drawbacks of Object-Oriented design and call and return architectures. Review section 5.3.**Resolved: 4/17/2020**

3. Eric: Make security contingencies in case of data leak/ security issues. There needs to be a consideration of the ramifications that using a database like MongoDB can have security wise due to the fact that it is cloud based. Perhaps creating our own local database that behaves similarly to a black box would be the most secure option. Review section 5.4.**Resolved: 4/16/2020**

4. Mike: Consider any architectural alternatives to the current design to ensure that the best implementation is chosen. Review section 5.5.**Resolved: 4/18/2020**

5. Mike: In section 5.1, what does it mean where it says "the log will be an array"?

   As each command for cruise control comes in, we will append it to a file as opposed to maintaining an array. This is not to say the log will not be treated like an object as each message will be a tuple of a time stamp and a string which will be appended to the log file. **Resolved: 5/4/2020**

6. Mike: Do we want to keep in section 5.2 the general definition of components in software?

   Yes, we do want to keep it as it helps guide understanding of how we will be using components. **Resolved: 5/4/2020**

7. Connie: In section 5.3, we should add a write_to_log() function.

   We added another function to our Control Management Diagram in Section 5.3 that would write each command to the log. **Resolved: 5/4/2020**

8. Connie: In section 5.3, we should add some accessor functions in our control management diagram.

   We added more functions to our Control Management Diagram in Section 5.3 that act as accessor methods. For example, we added: getSpeed(), and isCruiseControlOn() that return the current state of the vehicle. **Resolved: 5/4/2020**

9. Eric: In section 5.4, did we specify the specific programming language we will be using?

   We were deciding between implementing the Cruise Control System in Java or C++. We ultimately decided to use Java because the group as a whole feels more comfortable programming in this language. **Resolved: 5/4/2020**

10. Eric: In section 5.4, why are we using a database for local cruise control?

    Removal of the database as it will not be scalable so there is no reason to keep it. More so, since this is a local prototype and there is not a reason to connect the database. In order to store information, we have other means for it. It also leaves room for hackers to tamper with the software. **Resolved: 5/4/2020**

11. Hamzah: In section 5.4, there is a grammatical error in the first sentence of the last paragraph.

    This grammatical error was corrected. **Resolved: 5/4/2020**

12. Hamzah: In section 5.6, we need to specify where the issues were in section 5.

    The issues that were resolved in section 5.6 were updated to include the section where the issue was found. **Resolved: 5/4/2020**

13. Eric: The formatting of the charts in section 4.5 is off.

    The diagrams in section 4.5 were reformatted to correct mistakes that were made when uploading the diagram to the document. This will make comprehension of the diagrams much easier. **Resolved: 5/4/2020**

14. Connie: There is a spelling error in section 4.1.

    This spelling error was corrected. **Resolved: 5/4/2020**

6 Project Code

```
/**
 *  Authors: Eric Altenburg, Michael McCreesh, Hamzah Nizami, Constance Xu
 *  Description: Cruise Control Simulation for CS347
 *  Pledge: I pledge my honor that I have abided by the Stevens Honor System.
 */

import java.io.*;
import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.time.LocalTime;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.IOException;
import java.net.URL;
import javax.imageio.ImageIO;
import javax.swing.JPanel;
import javax.swing.*;


public class CruiseControl {

  /**
     *  Engine object holding value of car speed
     */
    public class Engine {
      // Data fields of engine
        private int current_speed;

        // Constructor
        public Engine(){
            current_speed = 0;
        }

        /**
       *  Returns current_speed
       */
```

```
41          public int get_current_speed(){
                return current_speed;
43          }

45          /**
           *  Sets current speed
47         */
            public boolean set_current_speed(int i) {
49            if (i >= 0 && i <= 160) {
                current_speed = i;
51              return true;
              } else {
53              return false;
              }
55          }
    }
57
  /**
59     *  Brake object holding value of brake being pressed
     */
61    public class Brake {
          // Keeps track if the brake is activated
63        private boolean is_brake;

65        // Constructor
          public Brake(){
67            is_brake = false;
          }
69
          /**
71        *  Returns value of brake
          */
73        public boolean is_brake_pressed(){
              return is_brake;
75        }

77        /**
          * Sets value of break then writes to log
79        */
          public void set_brake(boolean b, BufferedWriter
              ↪ cc_log, String inst) throws IOException{
81            is_brake = b;
              write_to_log(cc_log, inst);
83        }
```

```
85      }
87    /**
       * Log object holding time stamp and the instruction
89     **/
      public static class Log {
91      // Holds current instruction to be written
        private String instruction;
93
        // Constructor
95      public Log(String instruction) {
          this.instruction = instruction;
97      }

99      /**
           *  Returns a string of the and the instruction
101         */
        public String toString() {
103       LocalTime time = LocalTime.now();

105       return time + "\t\t" + instruction + "\n";
        }
107   }

109     // Data fields for cruise control
        private int cruise_speed;
111     private boolean is_activated;
        private boolean is_set;
113     private Engine cc_engine;
      private Brake cc_brake;
115
      // Constructor
117     public CruiseControl(){
          cc_engine = new Engine();
119       cc_brake = new Brake();
            cruise_speed = 0;
121         is_activated = false;
            is_set = false;
123     }

125     /**
         *  Writes to log through buffered writer and log
            ↪ object, throws IOException
127      *  if the files doesn't exist.
         */
```

```
129     public void write_to_log(BufferedWriter cc_log,
           ↪ String inst) throws IOException {
          // Creates log object with time stamp and action
             ↪ and append to cc_log
131     Log temp = new Log(inst);
        cc_log.append(temp.toString());
133     }

135     public int get_cruise_speed() {
          return cruise_speed;
137     }

139     /**
         * This function returns whether or not cruise
            ↪ control is activated
141      */
        public boolean is_cruise_control_activated(){
143         return is_activated;
        }
145
        /**
147      * This function returns whether or not cruise
            ↪ control is set
         */
149     public boolean is_cruise_control_set () {
          return is_set;
151     }

153     /**
         * This will decrement the cruise control speed by 1
            ↪ every time it is called.
155      * It will also ensure that is_activated = true and
            ↪ that the cruise control
         * speed is greater than 25. If it is 25 or lower, it
            ↪  will not decrement.
157      */
        public boolean decrement_speed(BufferedWriter cc_log)
           ↪  throws IOException {
159       // Check to see if the cc is activated and the
             ↪ speed is not at the limit of 25
          if (is_cruise_control_activated() && cruise_speed
             ↪  > 25) {
161         // If the car speed is equal to the pre cc
               ↪ decrement, then gas is not pressed and
               ↪ engine speed is updated
```

```
            if (cc_engine.get_current_speed() ==
                ↪ cruise_speed) {
163           cc_engine.set_current_speed(cc_engine.
                ↪ get_current_speed() - 1);
            }
165
            // Decrement cc speed and write to log then
                ↪ return true for successful decrement
167           cruise_speed -= 1;
            String decrement = "decremented CC speed by 1:
                ↪ " + cruise_speed;
169         write_to_log(cc_log, decrement);
            return true;
171       } else {
            // Write to log and return false for
                ↪ unsuccessful decrement
173         String failed = "Cruise control failed to
                ↪ decrement speed.";
            write_to_log(cc_log, failed);
175         return false;
        }
177   }

179   /**
       * This will increment the cruise control speed by 1
           ↪ every time it is called.
181    * It will also ensure that is_activated = true and
           ↪ that the cruise control
       * speed is actually set to something above or equal
           ↪ to 25 mph.
183    */
      public boolean increment_speed(BufferedWriter cc_log)
          ↪  throws IOException{
185     // Check to see if the cc is activated and the
            ↪ speed is not at the limits of 25 or 100
        if (is_cruise_control_activated() && cruise_speed
            ↪  >= 25 && cruise_speed < 100) {
187       // If the car speed is equal to pre cc
              ↪ increment, then gas is not pressed and
              ↪ engine speed is updated
          if (cc_engine.get_current_speed() ==
              ↪ cruise_speed) {
189         cc_engine.set_current_speed(cc_engine.
                ↪ get_current_speed() + 1);
          }
```

```
191
            // Increment cc speed and write to log then
                ↪ return true for successful increment
193             cruise_speed += 1;
                String increment = "incremented CC speed by 1
                    ↪ : " + cruise_speed;
195             write_to_log(cc_log, increment);
                return true;
197         } else {
                // Write to log and return false for
                    ↪ unsuccessful increment
199             String failed = "Cruise control failed to
                    ↪ increment speed.";
                write_to_log(cc_log, failed);
201             return false;
            }
203     }

205     /**
         * This function sets the cruise control speed as
             ↪ long as cruise control is activated
207      */
        public boolean set_speed(BufferedWriter cc_log)
            ↪ throws IOException {
209       // Cruise control cannot set speed unless car is
              ↪ going at least 25 mph or under 100 mph
            if (is_cruise_control_activated() && cc_engine.
                ↪ current_speed >= 25 && cc_engine.
                ↪ current_speed < 100) {
211             // Set cc speed to car speed write to log and
                    ↪ return true for successful set
                cruise_speed = cc_engine.get_current_speed();
213             String set_s = "Cruise control speed has been
                    ↪  set to " + cruise_speed;
                write_to_log(cc_log, set_s);
215             is_set = true;
                return true;
217         } else {
                // Write to log that it failed and return false
219             String failed = "Cruise control speed failed to
                    ↪  be set.";
                write_to_log(cc_log, failed);
221             return false;
            }
223     }
```

```
225     /**
        *  This function unsets the cruise control speed
           ↪ reverting it to 0
227      */
        public void unset_speed(BufferedWriter cc_log) throws
           ↪  IOException {
229       // Reset cruise speed and is_set to false
          cruise_speed = 0;
231       is_set = false;

233       // Write to log
          String unset_s = "Cruise control speed has been
             ↪ unset";
235       write_to_log(cc_log, unset_s);
        }
237
        /**
239      * This function will set is_actiaved to true to
           ↪ simulate turning on cruise control.
         */
241     public void activate_cruise_control(BufferedWriter
           ↪ cc_log) throws IOException {
          // Write to log and change value
243       String activation = "activated cruise control";
          write_to_log(cc_log, activation);
245         is_activated = true;
        }
247
        /**
249      * This function will set is_activated to false to
           ↪ simulate turning off cruise control.
         */
251     public void deactivate_cruise_control(BufferedWriter
           ↪ cc_log) throws IOException {
          // Write to log and change value
253       String deactivation = "deactivated cruise control";
          write_to_log(cc_log, deactivation);
255         is_activated = false;
        }
257
        public static void main (String[] args) throws
           ↪ IOException {
259       // Create log file to be passed around
```

```
    BufferedWriter cc_log = new BufferedWriter(new
        ↪ FileWriter("cc.log", true));
261 LocalTime time_of_car = LocalTime.now();
    cc_log.write(time_of_car + "\tBeginning of log:\n");
263
        // Create new cruise control object
265     CruiseControl j = new CruiseControl();

267     // Make new frame for buttons
        JFrame module = new JFrame("Cruise Control");
269
        // Various objects placed on frame
271     JLabel speedHere = new JLabel("Current Car Speed:
            ↪  ");
        speedHere.setBounds(40, 30, 150, 20);
273
        JLabel speedNotif = new JLabel("0 mph");
275     speedNotif.setBounds(200, 30, 350, 20);

277     JLabel ccSpeedHere = new JLabel("Current CC Speed
            ↪ : ");
        ccSpeedHere.setBounds(40, 90, 150, 20);
279
        JLabel ccSpeedNotif = new JLabel("0 mph");
281     ccSpeedNotif.setBounds(200, 90, 350, 20);

283     JLabel setCarSpeedHere = new JLabel("Set Car
            ↪ Speed: ");
        setCarSpeedHere.setBounds(40, 150, 150, 20);
285
        JTextField setCarSpeed = new JTextField();
287     setCarSpeed.setBounds(200, 150, 150, 20);

289     JButton returnToCC = new JButton("Return to CC
            ↪ Speed");
        returnToCC.setBounds(40, 210, 150, 20);
291
        JButton brake = new JButton("Brake");
293     brake.setBounds(200, 210, 150, 20);

295     JButton activatecc = new JButton("Activate CC");
        activatecc.setBounds(40, 270, 150, 20);
297
        JButton deactivatecc = new JButton("Deactivate CC
            ↪ ");
```

```
299         deactivatecc.setBounds(200, 270,150,20);

301         JButton setCCSpeed = new JButton("Set CC Speed");
            setCCSpeed.setBounds(40, 330, 150, 20);
303
            JButton unsetCCSpeed = new JButton("Unset CC
                ↪ Speed");
305         unsetCCSpeed.setBounds(200, 330, 150, 20);

307         JButton increasecc = new JButton("Increase CC
                ↪ Speed");
            increasecc.setBounds(40, 390, 150, 20);
309
            JButton decreasecc = new JButton("Decrease CC
                ↪ Speed");
311         decreasecc.setBounds(200, 390, 150, 20);

313         JLabel notifHere = new JLabel("Notifications: ");
            notifHere.setBounds(40, 450, 150, 20);
315
            JLabel notifications = new JLabel("Set a speed to
                ↪  turn on.");
317         notifications.setBounds(200, 450, 350, 20);

319         JLabel adminHere = new JLabel("Admin Login: ");
            adminHere.setBounds(40, 510, 150, 20);
321
            JPasswordField adminAccess = new JPasswordField()
                ↪ ;
323         adminAccess.setBounds(200, 510, 150, 20);


325
            //add all of the buttons to the frame.
327         module.add(returnToCC);
            module.add(brake);
329         module.add(deactivatecc);
            module.add(activatecc);
331         module.add(increasecc);
            module.add(setCarSpeedHere);
333         module.add(setCarSpeed);
            module.add(ccSpeedHere);
335         module.add(ccSpeedNotif);
            module.add(setCCSpeed);
337         module.add(unsetCCSpeed);
            module.add(decreasecc);
```

```
339         module.add(notifHere);
            module.add(notifications);
341         module.add(speedHere);
            module.add(speedNotif);
343         module.add(adminHere);
            module.add(adminAccess);
345
347         // Event to set the cc speed to whatever value
                ↪ the engine speed is.
            setCCSpeed.addActionListener(new ActionListener()
                ↪  {
349           public void actionPerformed(ActionEvent e) {
              try {
351             // Check to see if the cc is activated and
                    ↪ subsequently sets correctly
                  if(j.is_cruise_control_activated()){
353                 boolean temp = j.set_speed(cc_log);
                    if (temp) {
355                   // Change the colors of the
                          ↪ buttons to reflect that cc
                          ↪ is set
                      setCCSpeed.setForeground(Color.
                          ↪ GREEN);
357                   unsetCCSpeed.setForeground(
                          ↪ Color.RED);
                      module.add(setCCSpeed);
359                   module.add(unsetCCSpeed);

361                   // Update the cc speed notif
                          ↪ and push notification
                      notifications.setText("Cruise
                          ↪ control speed set.");
363                   ccSpeedNotif.setText(j.
                          ↪ get_cruise_speed() + "
                          ↪ mph");
                    } else {
365                   notifications.setText("CC Speed
                          ↪ failed to set.");
                    }
367               } else {
                    notifications.setText("Activate
                        ↪ cruise control to set speed
                        ↪ .");
369               }
```

```
            } catch (IOException f) {
              System.out.println(f.getMessage());
            }
          }
        });

        // Event that unsets the cc speed reseting the
           ↪ value to 0. Does not impact the engine
           ↪ speed.
        unsetCCSpeed.addActionListener(new ActionListener
           ↪ () {
          public void actionPerformed(ActionEvent e) {
            try {
              // Check to see if cc is activated
                    if(j.is_cruise_control_activated()){
                      // Check to see if cc is set as
                         ↪ nothing might have to be done
                      if (!j.is_cruise_control_set()) {
                        notifications.setText("Set CC
                           ↪ before unsetting it.");
                      } else {
                        // Unset the cc speed
                        j.unset_speed(cc_log);

                        // Change colors of buttons to
                           ↪ signify cc is unset
                          setCCSpeed.setForeground(Color.
                             ↪ RED);
                          unsetCCSpeed.setForeground(
                             ↪ Color.GREEN);
                          module.add(setCCSpeed);
                          module.add(unsetCCSpeed);

                          // Push the notification and
                             ↪ update cc speed notif
                          notifications.setText("Cruise
                             ↪ control speed unset.");
                          ccSpeedNotif.setText("0 mph");
                      }
                    } else {
                        notifications.setText("Activate
                           ↪ cruise control to unset
                           ↪ speed.");
                    }
            } catch (IOException f) {
```

```
403                 System.out.println(f.getMessage());
                }
405           }
          });
407
          // Event to set the car speed to whatever value
             ↪ is given. This only updates the engine
             ↪ speed and not
409       // the cc speed.
          setCarSpeed.addActionListener(new ActionListener
             ↪ () {
411           public void actionPerformed(ActionEvent e) {
                  try{
413                 // Grab current string in box and set
                       ↪ engine speed so long as >= 0 and
                       ↪ <= 160
                    String text_ = setCarSpeed.getText();
415                 if (j.cc_engine.set_current_speed(
                       ↪ Integer.parseInt(text_))) {
                      // Set the brake to false and push
                         ↪ to log
417                   String log_text = "Brake released.
                         ↪ Throttle engaged. Set speed
                         ↪ to: " + j.cc_engine.
                         ↪ get_current_speed() + " mph";
                      j.cc_brake.set_brake(false, cc_log,
                         ↪  log_text);
419
                      // Push the car speed notif,
                         ↪ regular notification, and
                         ↪ then reset the field
421                   notifications.setText("Car speed
                         ↪ set to " + text_ + " mph");
                      speedNotif.setText(text_ + " mph");
423                   setCarSpeed.setText("");
                    } else {
425                   notifications.setText("Failed to
                         ↪ set car speed.");
                    }
427               } catch (IOException f){
                    System.out.println(f.getMessage());
429               }
              }
431       });
```

```
433         // Admin login event. If the right password is
                ↪ given (Reza347), then a separate frame will
            // open up showing the logs.
435         adminAccess.addActionListener(new ActionListener
                ↪ () {
                public void actionPerformed(ActionEvent e){
437                 try{
                      // Grab the current password as char[]
439                   char[] text_ = adminAccess.
                          ↪ getPassword();
                      char[] pass = {'R', 'e', 'z', 'a', '3
                          ↪ ', '4', '7'};
441
                      // Compare to see if passwords match
443                   if(Arrays.equals(text_, pass)){
                        // Create new frame, panel, and
                            ↪ scrollpane for log
445                     JFrame admin = new JFrame("Admin:
                            ↪  Logs");
                        JPanel panel = new JPanel();
447                     JScrollPane scrollInfo=new
                            ↪ JScrollPane(panel,
                            ↪ JScrollPane.
                            ↪ VERTICAL_SCROLLBAR_ALWAYS,
                            ↪ JScrollPane.
                            ↪ HORIZONTAL_SCROLLBAR_ALWAYS
                            ↪ );

449                     // Update the log for the admin
                            ↪ accessing
                        j.write_to_log(cc_log, "ADMIN
                            ↪ ACCESSED.");
451
                        // Delete admin frame upon
                            ↪ closing window
453                     admin.addWindowListener(new java.
                            ↪ awt.event.WindowAdapter() {
                          @Override
455                       public void windowClosing(
                              ↪ java.awt.event.
                              ↪ WindowEvent windowEvent
                              ↪ ) {
                            admin.dispose();
457                       }
                        });
```

```
459
                        // Grab log file for JTextArea
                            ↪ and add it to panel
461                     JTextArea logs = new JTextArea();
                        FileReader reader = new
                            ↪ FileReader("cc.log");
463                     logs.read(reader, "cc.log");
                        panel.add(logs);
465
                        // Add scrollInfo to admin frame
                            ↪ then set size and
                            ↪ visibility
467                     admin.add(scrollInfo);
                        admin.setSize(650, 650);
469                     admin.setVisible(true);
                    } else {
471                     notifications.setText("Incorrect
                            ↪ password provided.");
                    }
473
                    // Reset the password for each
                        ↪ attempt either good or bad
                        ↪ entry
475                 adminAccess.setText("");
                } catch (IOException f){
477                 System.out.println(f.getMessage());
                }
479         }
        });
481
        // Event to increment the cc speed. If pressed,
            ↪ then the cc speed will increment by 1 and
            ↪ the car speed
483     // will only increment if it has the same speed.
        increasecc.addActionListener(new ActionListener()
            ↪  {
485          public void actionPerformed(ActionEvent e) {
                try {
487               // Check to see if cc is activated and
                      ↪ set for error notifs
                  if(j.is_cruise_control_activated()){
489                   if(j.is_cruise_control_set()){
                        // Check to see if the
                            ↪ increment was valid (i.e.
                            ↪  speed was at 25 or 100)
```

```
491                             if (j.increment_speed(cc_log))
                                    ↪ {
                                  // Push notification,
                                     ↪ updating the car speed
                                     ↪ notif and cc speed
                                     ↪ notif
493                                 notifications.setText("
                                     ↪ Incremented CC Speed
                                     ↪ to " + j.
                                     ↪ get_cruise_speed());
                                    speedNotif.setText(j.
                                     ↪ cc_engine.
                                     ↪ get_current_speed() +
                                     ↪  " mph");
495                                 ccSpeedNotif.setText(j.
                                     ↪ get_cruise_speed() +
                                     ↪ " mph");
                                } else {
497                                 notifications.setText("Failed
                                     ↪  to increment CC Speed.
                                     ↪ ");
                                }
499                             } else {
                                    notifications.setText("Set
                                     ↪ cruise control before
                                     ↪ increasing.");
501                             }
                            } else {
503                             notifications.setText("Activate
                                 ↪ cruise control before
                                 ↪ increasing.");
                            }
505                     } catch (IOException f){
                            System.out.println(f.getMessage());
507                     }
                    }
509         });

511         // Event to decrement the cc speed. If pressed,
               ↪ then the cc speed will decrease by 1 and
               ↪ the car speed
            // will only decrement if it has the same speed.
513         decreasecc.addActionListener(new ActionListener()
               ↪  {
              public void actionPerformed(ActionEvent e) {
```

```
            try {
              // Check to see if cc is activated and set
                 ↪ for error notifs
                    if(j.is_cruise_control_activated()){
                        if(j.is_cruise_control_activated
                           ↪ ()){
                          // Check to see if the
                             ↪ decrement was valid (i.e.
                             ↪  speed was at 25)
                          if (j.decrement_speed(cc_log))
                             ↪ {
                            // Push notification, update
                               ↪ the car speed, and the
                               ↪ cc speed
                            notifications.setText("
                               ↪ Decremented CC Speed to
                               ↪  " + j.get_cruise_speed
                               ↪ ());
                              speedNotif.setText(j.
                                 ↪ cc_engine.
                                 ↪ get_current_speed() +
                                 ↪  " mph");
                              ccSpeedNotif.setText(j.
                                 ↪ get_cruise_speed() +
                                 ↪ " mph");
                          } else {
                            notifications.setText("Failed
                               ↪  to decrement CC Speed.
                               ↪ ");
                          }
                        } else {
                            notifications.setText("Set
                               ↪ cruise control before
                               ↪ decreasing.");
                        }
                    } else {
                        notifications.setText("Activate
                           ↪ cruise control before
                           ↪ decreasing.");
                    }
            } catch (IOException f) {
              System.out.println(f.getMessage());
            }
          }
        });
```

```
539
        // Event to make sure the cc is activated.
           ↪ Nothing else changes.
541     activatecc.addActionListener(new ActionListener()
           ↪  {
            public void actionPerformed(ActionEvent e) {
543             try {
                  // If the cc is already activated, no
                     ↪ need to change things
545               if(j.is_cruise_control_activated()){
                    notifications.setText("Cruise
                       ↪ control already activated."
                       ↪ );
547               } else {
                    // Pass log to activate cc and
                       ↪ change colors for feedback
549                 j.activate_cruise_control(cc_log)
                       ↪ ;
                    activatecc.setForeground(Color.
                       ↪ GREEN);
551                 deactivatecc.setForeground(Color.
                       ↪ RED);
                    module.add(activatecc);
553                 module.add(deactivatecc);

555                 // Push notification
                    notifications.setText("Cruise
                       ↪ Control activated");
557               }
                }
559             catch(IOException f) {
                    System.out.println(f.getMessage());
561             }
            }
563     });

565     // Event to deactivate the cc. Upon being pressed
           ↪ , it will unset current cc setting to 0.
        deactivatecc.addActionListener(new ActionListener
           ↪ () {
567      public void actionPerformed(ActionEvent e) {
            try {
569           // Check to see if cc is deactivated
              if (!j.is_cruise_control_activated()) {
```

```
571                     // Already deactivated so no need to
                           ↪ change colors just say it's
                           ↪ already deactivated
                        notifications.setText("Cruise control
                           ↪  already deactivated");
573                   } else {
                        // Pass log and deactivate cc and
                           ↪ change colors
575                     j.deactivate_cruise_control(cc_log);
                          activatecc.setForeground(Color.RED)
                             ↪ ;
577                       deactivatecc.setForeground(Color.
                             ↪ GREEN);
                          module.add(activatecc);
579                       module.add(deactivatecc);

581                       // If the cruise control is set we
                             ↪ want to change the colors and
                             ↪  add them to the frame
                          if (j.is_cruise_control_set()) {
583                         j.unset_speed(cc_log);
                            setCCSpeed.setForeground(Color.
                               ↪ RED);
585                         unsetCCSpeed.setForeground(Color.
                               ↪ GREEN);
                  module.add(setCCSpeed);
587                         module.add(unsetCCSpeed);

589                         // Push the cc notif as the speed
                               ↪  is now 0
                            ccSpeedNotif.setText(j.
                               ↪ get_cruise_speed() + " mph"
                               ↪ );
591                       }

593                       // Push the notification saying it
                             ↪ was deactivated
                          notifications.setText("Cruise
                             ↪ control deactivated");
595                     }
                    } catch (IOException f) {
597                   System.out.println(f.getMessage());
                    }
599               }
          });
```

```
601
        // Event to have the car speed return to the
           ↪ cruise control speed. Ex: Car speed gas
           ↪ pressed and
603     // speed is now 90 but cc is set to 65. Upon
           ↪ being pressed, the car speed will revert to
           ↪  cc speed
        returnToCC.addActionListener(new ActionListener()
           ↪  {
605       public void actionPerformed(ActionEvent e) {
            // Check to see if set
607       if (j.is_cruise_control_set()) {
            // Have engine speed become current set cc
               ↪ speed
609         j.cc_engine.set_current_speed(j.
               ↪ get_cruise_speed());

611         // Update the car speed notif and push notif
            speedNotif.setText(j.cc_engine.
               ↪ get_current_speed() + " mph");
613         notifications.setText("Car speed returned to
               ↪ CC speed.");
          } else {
615         notifications.setText("CC must be set before
               ↪ returning.");
          }
617       }
        });
619
        // Event when the brake is pressed. Upon being
           ↪ pressed, it should set speed vals to 0
621     // and unset cc speed but not deactivate.
        brake.addActionListener(new ActionListener() {
623         public void actionPerformed(ActionEvent e){
              try {
625             // Bounds to check if brake is
                   ↪ activated during cruise control
                   ↪ set
                  if(j.is_cruise_control_set()){
627                 // Update colors and add to frame
                    setCCSpeed.setForeground(Color.
                       ↪ RED);
629                 unsetCCSpeed.setForeground(Color.
                       ↪ GREEN);
                    module.add(setCCSpeed);
```

```
631                         module.add(unsetCCSpeed);

633                         // Set engine speed to 0, push to
                                ↪  log, set brake to true,
                                ↪ and unset cc
                            j.cc_engine.set_current_speed(0);
635                         String log_text = "Brake
                                ↪ activated. CC Speed set to
                                ↪ 0. Car speed set to " + j.
                                ↪ cc_engine.get_current_speed
                                ↪ () + " mph";
                            j.cc_brake.set_brake(true, cc_log
                                ↪ , log_text);
637                         j.unset_speed(cc_log);

639                         // Push the notification, and
                                ↪ update car and cc speed
                                ↪ notif
                            notifications.setText("Brake
                                ↪ pressed. Car speed: " + j.
                                ↪ cc_engine.get_current_speed
                                ↪ () + " mph");
641                         speedNotif.setText(j.cc_engine.
                                ↪ get_current_speed() + " mph
                                ↪ ");
                            ccSpeedNotif.setText(j.
                                ↪ get_cruise_speed() + " mph"
                                ↪ );
643                     } else {
                          // If not set, just change engine
                             ↪ speed to 0, add log text, and
                             ↪  set brake to true.
645                       j.cc_engine.set_current_speed(0);
                          String log_text = "Brake
                                ↪ activated. CC Speed set to
                                ↪ 0. Car speed set to " + j.
                                ↪ cc_engine.get_current_speed
                                ↪ () + " mph";
647                       j.cc_brake.set_brake(true, cc_log
                                ↪ , log_text);

649                         // Push notification and update
                                ↪ only car speed as cc not
                                ↪ set
```

```
                        notifications.setText("Brake
                            ↪ pressed. Car speed: " + j.
                            ↪ cc_engine.get_current_speed
                            ↪ () + " mph");
                        speedNotif.setText(j.cc_engine.
                            ↪ get_current_speed() + " mph
                            ↪ ");
                    }
                } catch (IOException f){
                    System.out.println(f.getMessage());
                }
            }
        });

        // Event to close window and then flush the
            ↪ stream
        module.addWindowListener(new java.awt.event.
            ↪ WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.
                ↪ WindowEvent windowEvent) {
              // check to see if the user wants to close
                  ↪ out the window
                if (JOptionPane.showConfirmDialog(module,
                    ↪  "Are you sure you want to close
                    ↪ this window?", "Close Window?",
                    ↪ JOptionPane.YES_NO_OPTION,
                    ↪ JOptionPane.QUESTION_MESSAGE) ==
                    ↪ JOptionPane.YES_OPTION) {
                        try{
                          // Append new line to
                              ↪ distinguish different
                              ↪ drives and close stream
                          cc_log.append("\n");
                            cc_log.close();
                            System.exit(0);
                        } catch (IOException f) {
                            System.out.println(f.
                                ↪ getMessage());
                        }
                }
            }
        });

```

```
        // Setting module frame to be visible
679     module.setSize(500,600);
        module.setLayout(null);
681     module.setVisible(true);

683   }
}
```

7 Issues

1. Issue determining which Java GUI framework to use. **Resolved on 05/07/2020. Decided to use Java Swing due to it's implementation relying on pure Java FX, which allows for greater portability. Furthermore, Java Swing has a greater support network that would enable faster remediations.**

2. Objects would not properly update in the GUI. **Resolved on 05/08/2020. Was an issue about how the objects were not declared in scope.**

3. Brake did not stop cruise control. **Resolved on 05/09/2020. Ensured that brake unsets cruise control yet keeps it activated so that the user can go back to enabling the cruise control by resetting the speed.**

4. Increment Cruise Control button incorrectly updated the engine speed instead of cruise control speed. **Resolved on 05/09/2020. Fixed the increment cruise control button to update the cruise control speed and not the engine speed.**

5. Able to set a cruise control speed before activation. **Resolved on 05/09/2020. Ensured that set cruise control can only be used as long as the cruise control system is activated by using a boolean flag.**

6. Log is not properly updated with all cruise control information. **Resolved on 05/10/2020. Ensured that after each cruise control activity the log is written with a BufferedWriter.**

7. Log has too much information by including all the activities that the car is doing. **Resolved on 05/10/2020. Remediated the issue**

by only allowing the log to be written when cruise control is activated.

8. When accessing the admin logs in the GUI, if there is too much information in the window you cannot scroll down to view all of it. Need to implement scrolling functionality into the admin JFrame. **Resolved on 05/11/2020. Implemented a correct JScrollPane.**

9. When exiting out of the admin pane, the entire system shuts down. **Resolved on 05/11/2020. Deleted the call to System.exit when deleting out of admin pane and replaced it with a simple call to delete the frame and continue running cruise control.**

10. Visual notifcations for the program does not include changes in cruise control activation and deactivation. **Resolved on 05/11/2020. Extended the notifications to handle cruise control activation and deactivation.**

11. Scheduling conflicts due to Final Exam Week made team meetings less frequent then desired. **Resolved on 05/11/2020. Adopted a model where the group would meet for an hour every few days to discuss things to fix. Each individual member would then take a task and then fix it. On 05/11/2020, the group was able to devote several hours into refining the entire document and presentation.**