

CSC216: Project 2

Project 2, Part 1: Wally's Service Garage

[Home](#) > [Projects](#) > [Project 2](#) > [Project 2, Part 1: Wally's Service Garage](#)

[Project 2, Part 2: Wolf Travel Tours](#) ►

Project 2, Part 1: Wally's Service Garage

Project 2 requires you to go through standard software development phases to design, implement, and test a complete Java program consisting of multiple source code files and JUnit test cases. The project comes in two parts. Deliverables for Part 1 are:

1. Design document that includes a UML class diagram.
2. Black box test plan.

Part 1 Due Date: Friday, July 5 at 11:45 PM

Late Deadline (BBTP): Sunday, July 7 at 9:00 AM

Project 2, Part 1 MUST be completed individually

Problem Overview

Wally's Service Garage is a new service garage that services all kinds of cars and light trucks. The shop has plans for 30 service bays and almost as many mechanics to staff them. However, scheduling which vehicles go where and when they go is no small task. That is the primary job of the service manager.

Wally's offers special paid "tier" plans whereby customers can get quicker service on their vehicles. There are three tiers: Silver, Gold, and Platinum. Platinum tier vehicles get the fastest service, followed by Gold, and then by Silver. Vehicles with no tier plans get the slowest service. So theoretically, all Platinum plan vehicles are serviced before all Gold plan vehicles; all Gold plan vehicles are

served before all Silver plan vehicles; all Silver plan vehicles are serviced before all vehicles with no plan. The only exception to this is for hybrid or electric vehicles, which may be serviced before others according to the restrictions on some service bays.

Your company is bidding on contract to build software to help the service manager schedule and manage the repair waiting lot and the service bays. To sweeten your company's bid, your boss has asked you to create a simple prototype to illustrate how the software might work. The prototype must support the current operational procedures and restrictions.

Figure 1 shows a sample prototype GUI for the Auto Repair Shop software bid. The GUI must provide all of the functionality indicated in the figure.

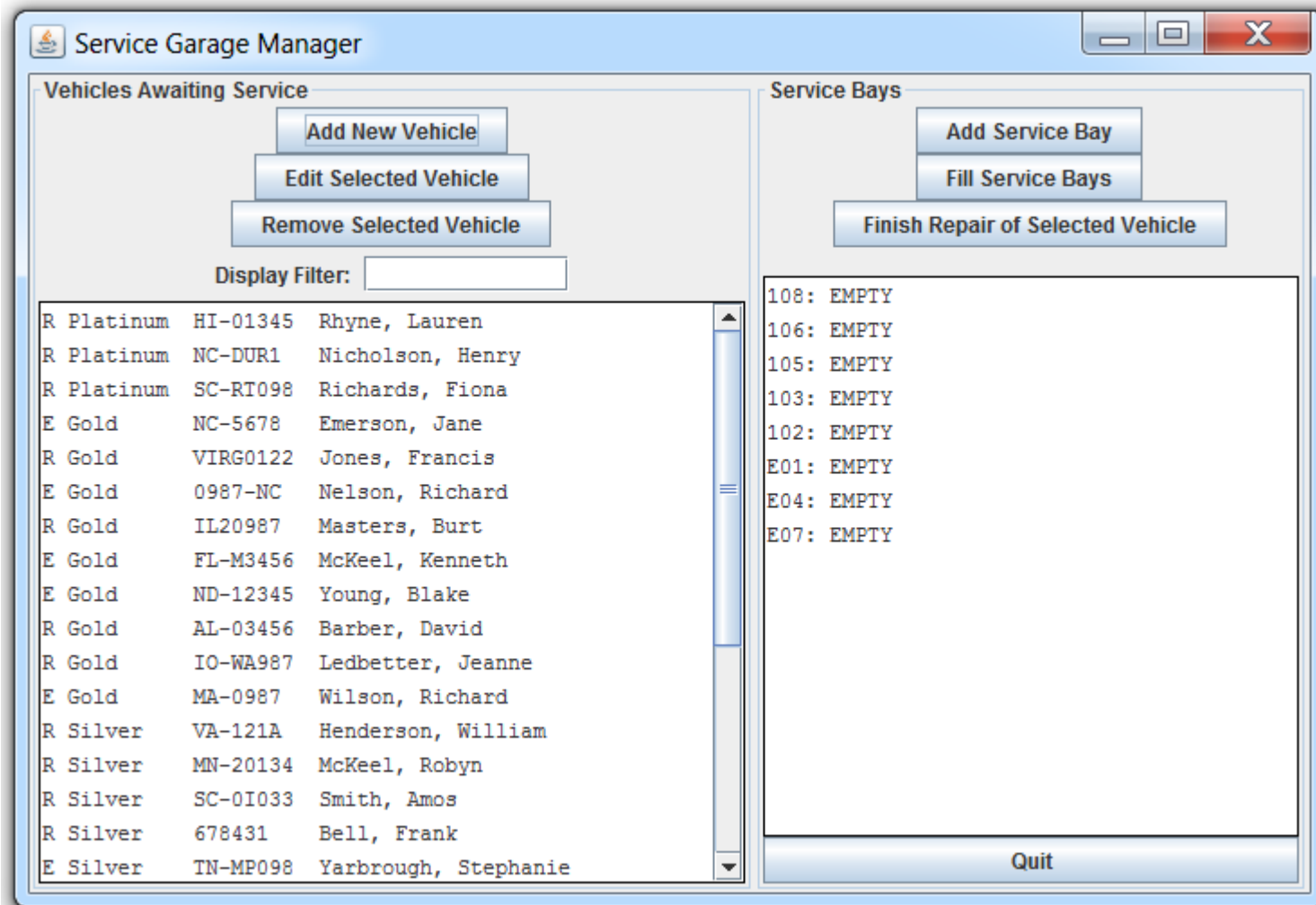


Figure 1: Wally's Service Garage service manager prototype GUI.

Vehicle service management guidelines and restrictions

The service manager keeps information the vehicles waiting to be serviced in a Vehicles Awaiting Service list (list), following these guidelines and restrictions:

1. In an effort to increase the sales of hybrid and electric vehicles, Wally's dedicates at least 1/3 of its stations to their repair and maintenance. A service bay dedicated to hybrid or electric vehicles is called an HEV-bay. Hybrid and electric vehicles can be serviced in regular service bays, as well. Regular vehicles, however, cannot be serviced at HEV-bays.
2. When an owner brings in a vehicle for service, the service manager creates an entry in the Vehicles Awaiting Service list that includes the owner name, the service plan, and whether the vehicle is regular or hybrid/electric. **Figure 2 below shows entry for a hybrid vehicle with a Gold tier service plan.**
3. The Vehicles Awaiting Service list is ordered first according to service plan tier, then according to when the vehicle is entered into the Vehicles Awaiting Service list (typically arrival time). In other words, all vehicles with Gold tier status appear on the Vehicles Awaiting Service list before those with Silver tier status, and Silver tier status vehicles appear before vehicles with no paid tier status. For each status level, the first vehicle to be assigned that level is listed before the second to be assigned that level, and so on.
4. The service manager can arbitrarily change the service plan tier on any vehicle in the Vehicles Awaiting Service list, thus changing the position of the vehicle in the Vehicles Awaiting Service list. A vehicle reassigned a tier status is moved immediately behind all other vehicles with equal or higher tier status.
5. Vehicles are serviced in order of where they appear on the Vehicles Awaiting Service list, with the exception of point 6 below.
6. If only HEV-bays are available when a regular vehicle is at the front of the Vehicles Awaiting Service list, then hybrid/electric vehicles further back in the Vehicles Awaiting Service list can jump ahead of the regular vehicles in order to fill the empty service bays.
7. When a vehicle goes to a service bay, it is removed entirely from the waiting list.

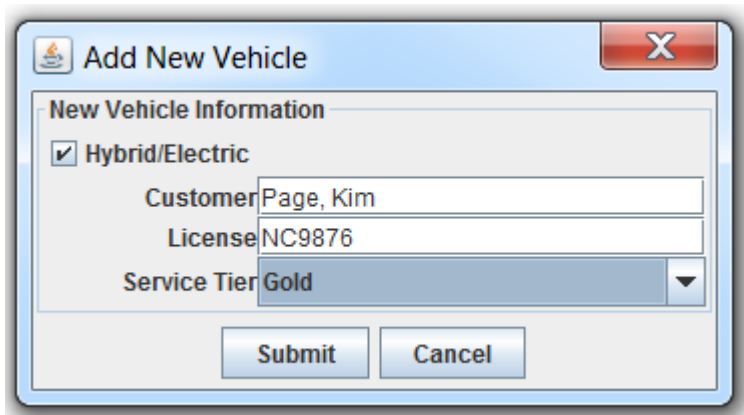


Figure 2: Adding a vehicle to the Vehicles Awaiting Service area

Service bay guidelines

The garage's service bays constitute a list of Service Bays that is independent of the list of Vehicles Awaiting Service. Service bay restrictions are as follows:

1. Service bays must be open to be able to service vehicles. No more than 30 service bays can be open at any time.
2. Since Wally's dedicates 1/3 of the service bays to hybrid or electric vehicles, at least 1/3 of the open stations are HEV-bays.
3. The garage always starts each day by opening exactly 8 service bays. The first 8 service bays are numbered as follows: 108, 106, 105, 103, 102, E01, E04, E07.
4. Service bays with prefix "E" are HEV-bays.
5. The sequence of numbers 01, 02, 03, ... 08 indicates the order in which the first 8 service bays are opened (as explained next).
6. If fewer than 30 service bays are open, the service manager can open another.
 - If the newly opened service bay is an HEV-bay, it is added to the end of the list. Its number is of the form:
E<next number in sequence>
 - If the newly opened service bay can service any kind of vehicle (an regular service bay), it is added to the front of the list. Its number is of the form:
1<next number in sequence>
7. A newly opened service bay is automatically a general purpose service bay if at least 1/3 of the total currently-open stations are HEV-bays after the addition. Otherwise, the the newly opened service bay is automatically an HEV-bay.

Service Manager operations

The service manager is responsible for handling the vehicles awaiting service and the service bays. The software must enable the service manager to do the following:

1. Admit a new vehicle for repair service. Vehicles are characterized by type (regular vs hybrid/electric), service tier, owner name, and license. See Figure 2 above.
2. Remove a vehicle from the Vehicles Awaiting Service list. In this case, the owner is no longer seeking repair service for the vehicle.
3. Set the tier of a vehicle after the vehicle is already on the Vehicles Awaiting Service list.
4. Open a new service bay.
5. Fill the empty service bays with vehicles in the Vehicles Awaiting Service list.
6. Remove a vehicle from a service bay, thus making that service bay empty (and subsequently able to service other vehicles). See Figure 1 above.

The [Manageable.java](#) interface describes behaviors for the garage service manager. The behaviors are detailed in the use case section below.

User interface displays

The user interface must display the service bay numbers and current assignments (EMPTY, or **license plate with owner name**). It must also display the list of vehicles in order of their positions in the Vehicles Awaiting Service list. Since that list can be quite long, the user must be able to filter the display to show only vehicles whose owner last names match a prefix filter. Figure 3 shows a filtered list. Only owners (in the Vehicles Awaiting Service list) whose last names start with “H” (ignoring case) are displayed. The user can clear the text field to clear the filter and once again display all vehicles.

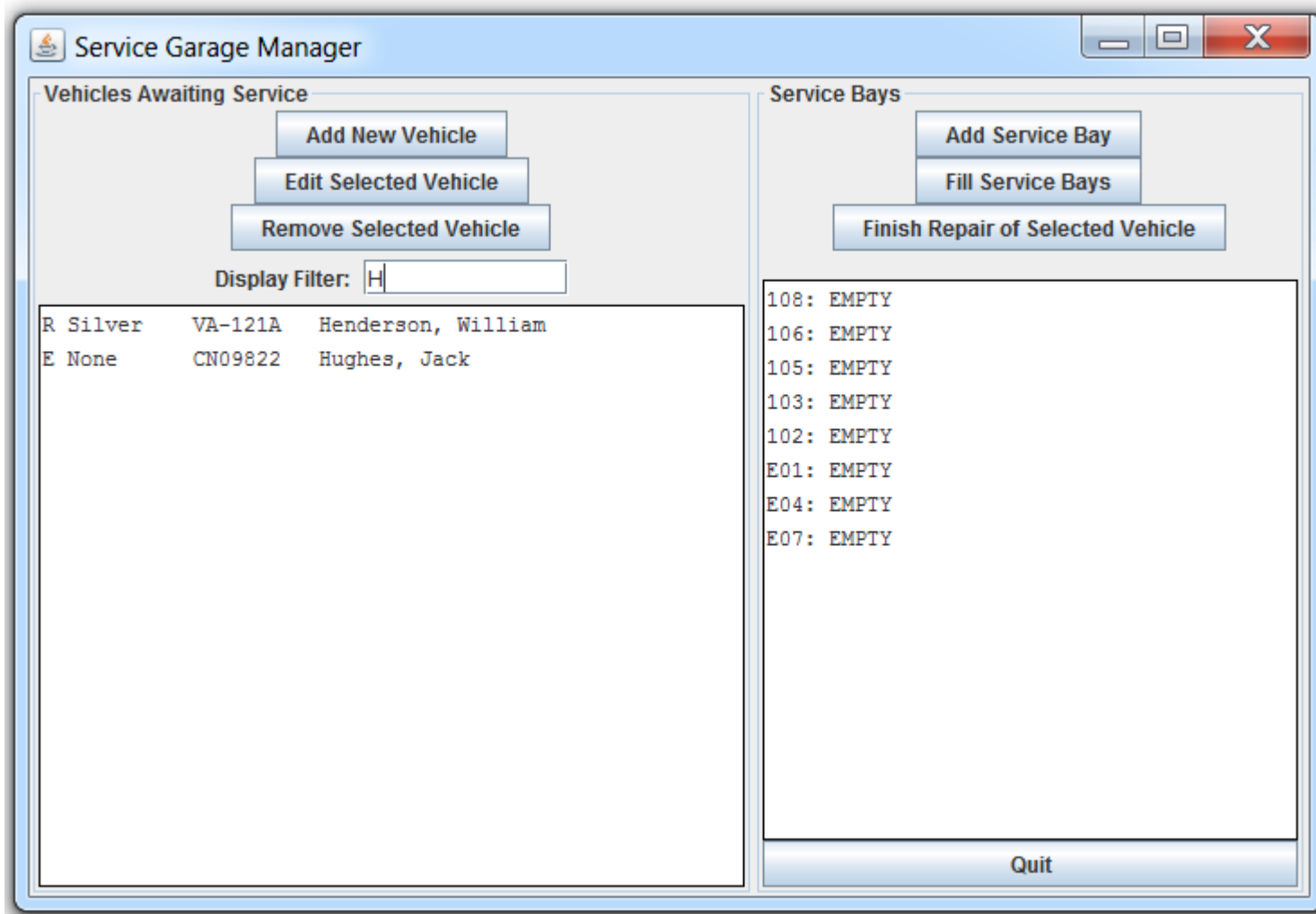


Figure 3: Filtering the list of Vehicles Awaiting Service with “H” prefix.

Input File Specifications

Your software must accept as input a file (for example, [cars.txt](#)) that contains information about vehicles waiting to be serviced. Each line in the input file corresponds to a vehicle, formatted as follows:

* is one of the characters 'R' or 'E' to indicate (R)egular or hybrid/(E)lectric vehicle. * is an integer indicating vehicle tier. 0 = no tier, 1 = Silver, 2 = Gold, 3 = Platinum * is a string containing no interior whitespace characters representing the car license. The license must can contain a sequence of at most 8 valid non-whitespace characters. (_For example, "AB 123" is invalid because it contains a space between 'B' and '1'. " AB-123 " is valid, because it contains a sequence of less than or equal to 8 non-whitespace characters after the leading and trailing whitespace is removed.) * is a string with at least one non-whitespace character indicating the owner's name.

Use Case 1: Startup

Preconditions: A valid vehicle information file is accessible.

Main Flow: The user starts the program with a command-line argument for the name of the file representing the vehicle information [S1]. If there is no command-line argument, the program opens a file chooser for the user to select the vehicle information file. The program builds the list of vehicles awaiting service [S2] and opens the GUI display with the list of vehicles awaiting service and list of open service bay information [S3].

Subflows:

[S1] The user specifies the file as a command-line argument or through a file chooser [S2][E1].

[S2] The list of vehicles awaiting service is created by processing each line of the file as described above [S3]. Any line that fails to meet the specified format is ignored.

[S3] The display opens, displaying the vehicles in the Vehicles Awaiting Service list ordered first by tier and second by position in the file. The display of each vehicle is set with the following field formats:

1. First 2 characters: 'R' or 'E' followed by blank
2. Next 10 characters: Service Tier, left justified in a field of width 10
3. Next 10 characters: License, left justified in a field of width 10
4. Remaining characters: Owner name

[S4]The GUI lists 8 service bays in its Service Bay list (of open service bays), each empty. Those first 8 have IDs as follows: 108, 106, 105, 103, 102, E01, E04, E07. An ID that begins with '1' denotes a regular service bay. An ID that begins with 'E' denotes an hybrid/electric service bay. All IDs consist of exactly 3 characters (a prefix and two digits). See Figure 5.

Alternative Flows:

[E1] If the filename specified by the user does not exist, the program exits with an error dialog (See Figure 4).

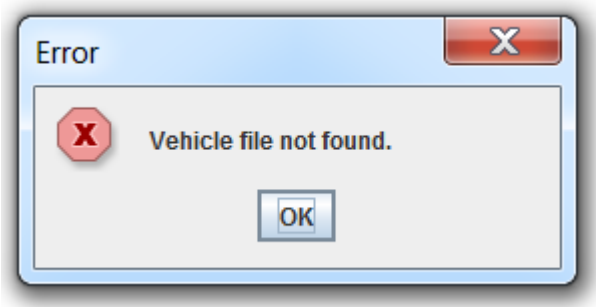


Figure 4: Error dialog for vehicle file not found.

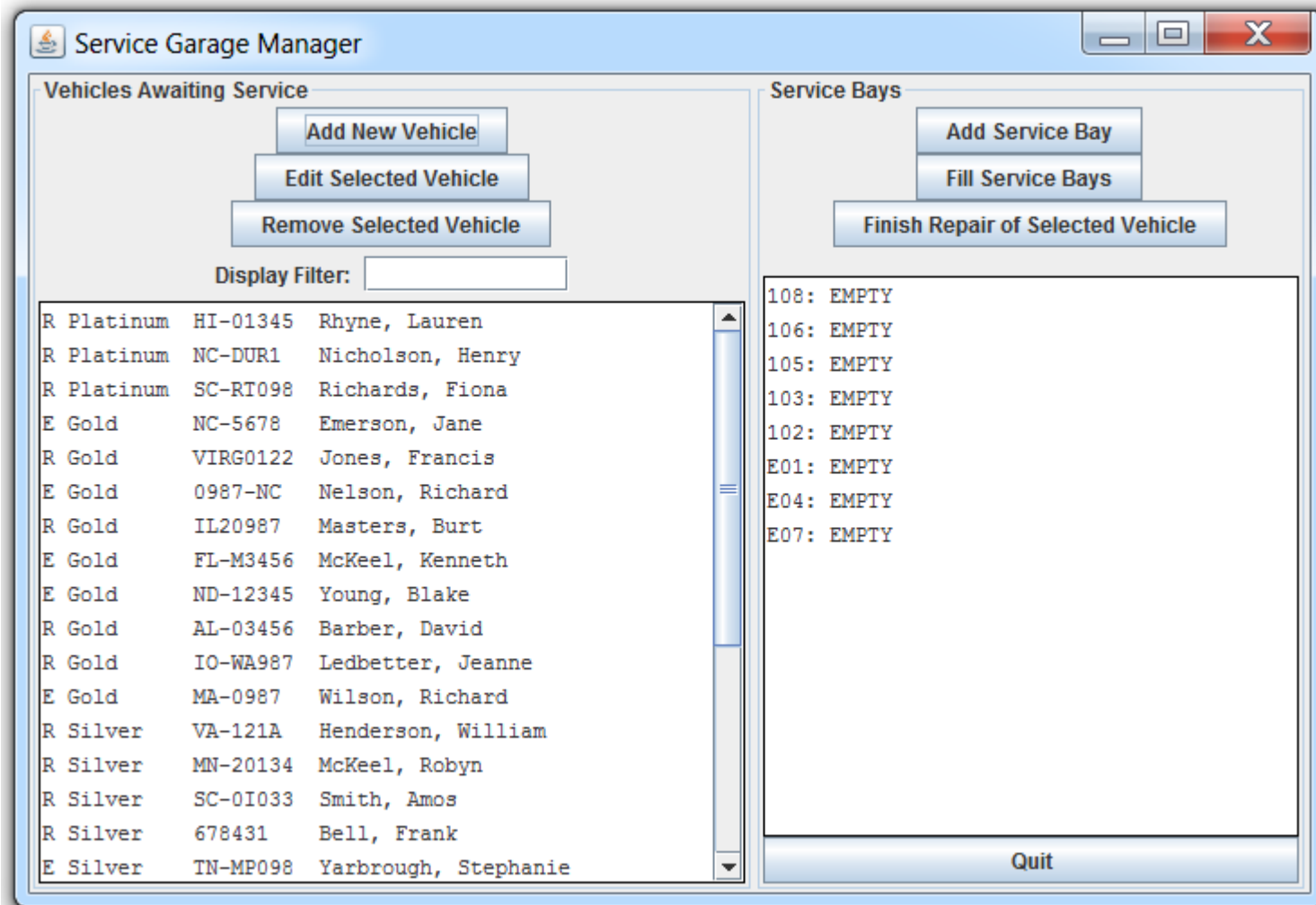


Figure 5. Initial service bay display.

Use Case 2: Add a vehicle to the Vehicles Awaiting Service list

Precondition: The main GUI window is open.

Main Flow: The user clicks Add New Vehicle. A dialog opens with widgets for data entry. The user enters information, then clicks Submit. The vehicle is added according to the specified vehicle data. The user can alternatively click Cancel.

[S1] A checkbox for Hybrid/Electric is *un-checked* by default. A combo (dropdown) for tier defaults to *None*. There are textfields for license [E2], and owner name [E3]. The user enters/selects all these data, which are sufficient to construct a new vehicle.

[S2] The user clicks *Submit* to add the new vehicle to the wait list.

[S3] The user data are changed to trim whitespace off the beginning and ends of the license and owner names.

[S4] The wait list display shows the new vehicle at the end of the sublist of vehicles with tiers that are at least as high as the newly added one. The format for each vehicle follows that described in [UC1, S3].

Alternative Flows:

[E1] If the user clicks *Cancel*, the new vehicle dialog closes and the wait list remains unchanged.

[E2] If the user clicks *Submit* but the owner name (customer) field is blank or all whitespace, a warning opens with the message "Owner name cannot be blank." See Figure 6. The new vehicle dialog remains open.

[E3] If the user clicks *Submit* but the license field is blank or all whitespace, a warning opens with the message "License cannot be blank." See Figure 7. The new vehicle dialog remains open.

[E4] If the user clicks *Submit* but the license field (after being trimmed of leading and trailing whitespace) contains a sequence of more than 8 non-whitespace characters, a warning opens with the message "License cannot be more than 8 characters." See Figure 8. The new vehicle dialog remains open.

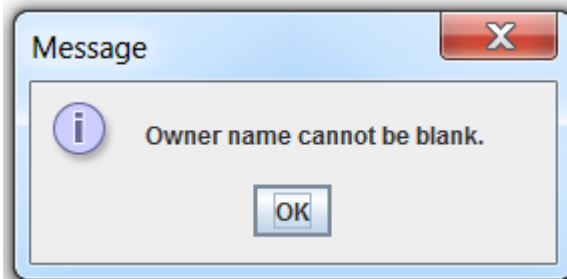


Figure 6: Invalid owner error message.

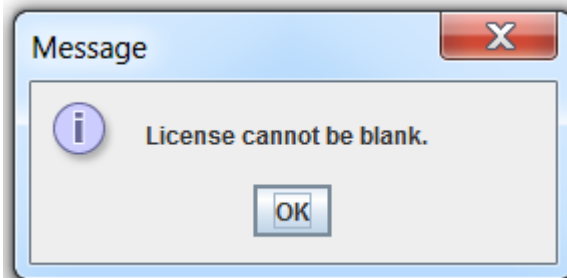


Figure 7: Empty license error message.

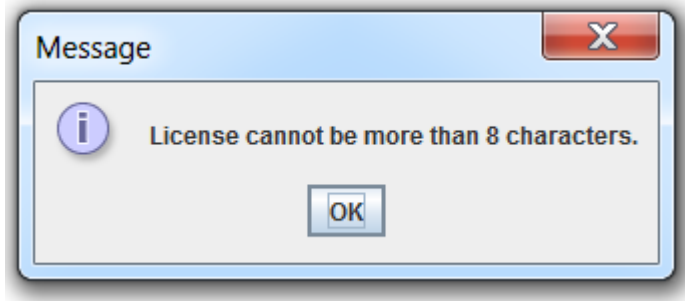


Figure 8: Invalid license length error message.

Use Case 3: Remove a vehicle from the Vehicles Awaiting Service list

_Precondition: _The main GUI window is open. There is at least one vehicle in the Vehicles Awaiting Service list.

_Main Flow: _The user selects a vehicle from the Vehicles Awaiting Service list and clicks “Remove Selected Vehicle” [E1]. The selected vehicle is removed from the awaiting service list and the GUI refreshes the list to display only the vehicles remaining in the list.

Alternate Flows:

[E1]. If no vehicle is selected or the selection does not correspond to an actual vehicle (index is incorrect), then no vehicle is removed.

Use Case 4: Re-assign the tier for a vehicle

_Precondition: _The main GUI window is open. There is at least one vehicle in the Vehicles Awaiting Service list.

Main Flow: _The user selects a vehicle from the Vehicles Awaiting Service list, then selects “Edit Selected Vehicle”. The user updates the tier change from the corresponding tier selections (see Figure 9), then clicks _Submit[E1][E2]. The selected vehicle’s tier updates according the new value chosen. The GUI refreshes the Vehicles Awaiting Service list to reflect the new list ordering resulting from the change.

Alternate Flows:

[E1]. If no tier is chosen, the tier value shall default to “None”.

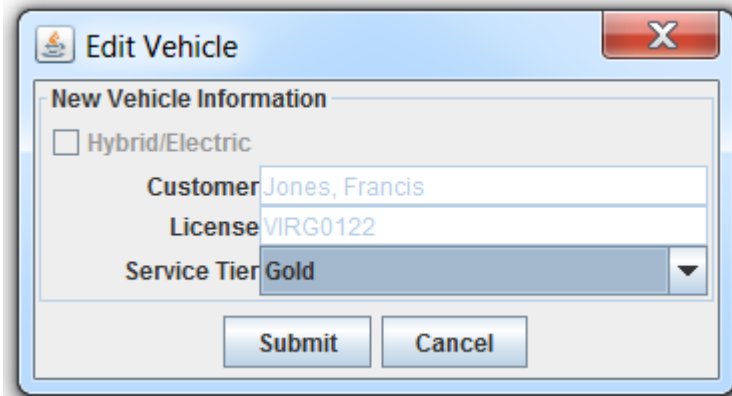


Figure 9: Update tier window.

Use Case 5: Filter the display list

Precondition: The main GUI window is open.

****Main Flow:**** The user enters a prefix string in the *Display Filter* text field. The GUI changes the Vehicles Awaiting Service list display to show only vehicles whose owner's last name start with the prefix string (ignoring case). See Figure 3.

Subflows

[S1] The user types a prefix string in the Display Filter text field and presses *Enter/Return* [E1].

[S2] The GUI displays all vehicles whose owners' names begin with the prefix. The match is not case sensitive; leading and trailing blanks in the filter are ignored.

Alternative Flows

[E1] If the prefix string is blank, null, or all whitespace, the GUI shows all vehicles on the wait list.

Use Case 6: Add service bay

Precondition: The main GUI window is open.

****Main Flow:**** The user clicks *Add service bay*. A new service bay is added to the Service Bays list [S1][S2][S3][E1]. The Service Bay list is refreshed, with the new service bay is shown as EMPTY.

Subflows:

[S1] If a new service bay would make the number of hybrid/electric service bays less than $\frac{1}{3}$ of the total number of service bays, the new service bay is added as a hybrid/electric service bay to the end of the sublist of hybrid/electric service bays.

[S2] If a new service bay would keep the number of hybrid/electric service bays at least $\frac{1}{3}$ of the total number of service bays, the new service bay is added as a regular service bay to the beginning of the sublist of regular service bays.

[S3] The new service bay number is "1" (for regular service bay) or "E" (for hybrid/electric service bay) plus the new service bay's number. The suffix is 1 + the previous largest service bay number (where "+" here means integer addition rather than string concatenation).

Alternative Flow:

[E1] If there are already 30 total service bays, no new service bay is added.

Use Case 7: Fill service bays

Precondition: The main GUI window is open.

_Main Flow: _The user clicks "Fill Service Bays": Vehicles from the Vehicles Awaiting Service list fill the empty service bays according to their position on the list and type of vehicle (regular or hybrid/electric) [S1][S2].

Subflows:

[S1] The system attempts to assign Vehicles Awaiting Service to available service bays. This process starts one-vehicle-at-a-time at the front of the Vehicles Awaiting Service list (containing vehicles with the highest tiers) and continues toward the end of the list until there are no empty service bays left appropriate for the remaining vehicles.

Regular vehicles always seek empty service bays starting at the beginning of the list of service bays.

Hybrid/electric vehicles always seek empty service bays starting at the end (where the HEV-bays are listed).

Regular vehicles cannot pick HEV-bays. Hybrid/electric vehicles can pick HEV-bays *or* regular service bays.

If only HEV-bays are empty, the first hybrid/electric vehicles in the list (which may not necessarily be at the front of the list) fill the HEV-bays.

[S2] Every vehicle that is successful at finding an appropriate service bay is removed from the list of Vehicles Awaiting Service and entered into the selected service bay.

[S3] The GUI refreshes the Vehicles Awaiting Service list, which now does not contain any vehicles that successfully entered a service bay.

[S4] The GUI also refreshes the Service Bay list, formatting the display as follows:

1. Bay ID ("E" or "1" prefix and number) followed by a colon and blank.
2. License left justified in a field of width 9 if the bay is occupied, or EMPTY if the bay is empty.
3. Owner name if the bay is occupied.

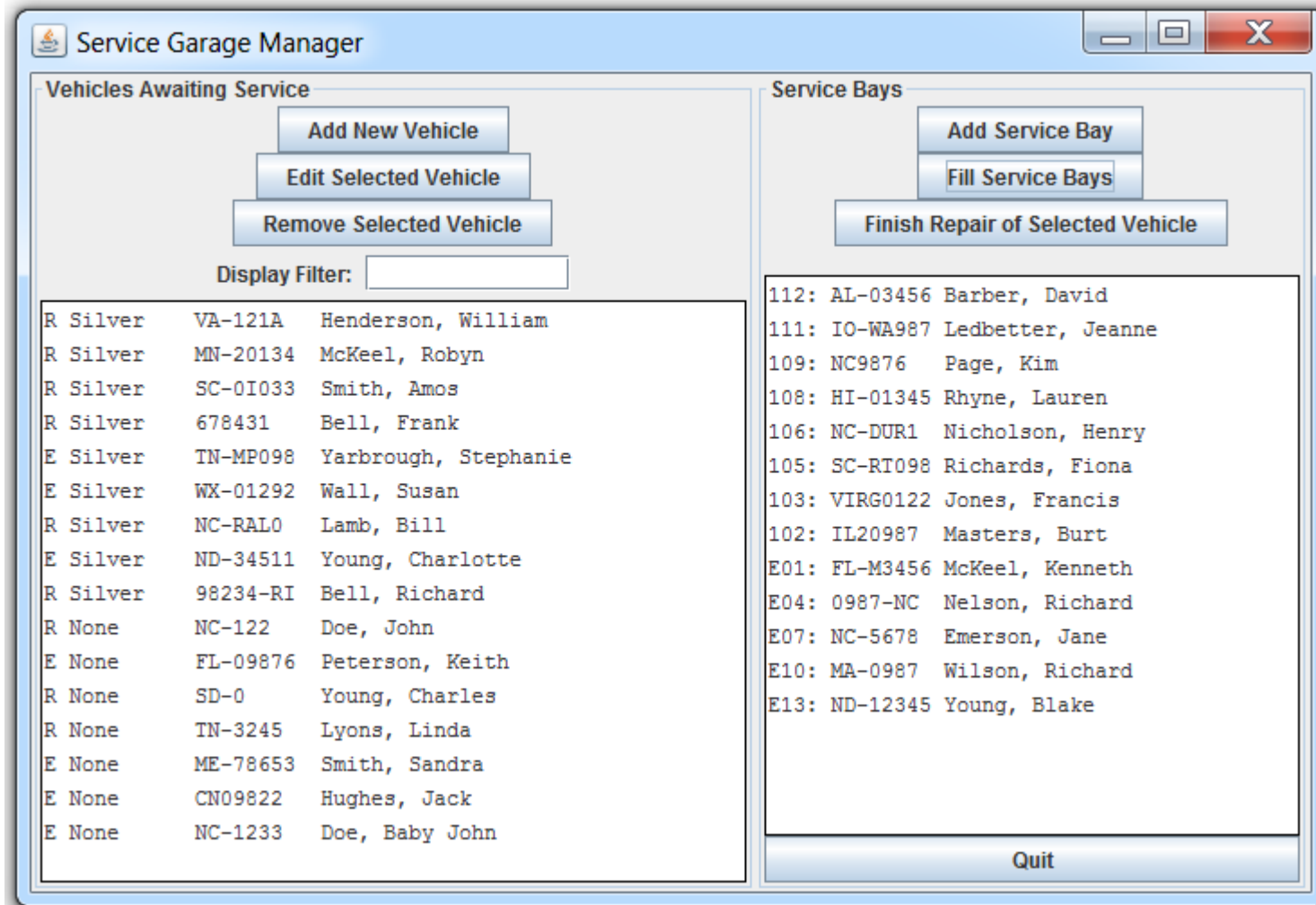


Figure 10: Vehicles Awaiting Service are automatically assigned to an appropriate service bay after clicking "Fill Service Bays".

Use Case 8: Finish vehicle repair

****Precondition:**** The main GUI window is open.

Main Flow: _The user selects a service bay from the Service Bays list and clicks “Finish Repair of Selected Vehicle” [E1]. The vehicle in the selected service bay is released, leaving the service bay empty.

Subflows:

[S1] The user selects a bay from the Service Bay list and clicks *Finish Repair of Selected Vehicle*.

[S2] The vehicle in the selected bay is removed, leaving the bay empty.

[S3] The GUI refreshes the Service Bay list to show EMPTY beside the number of the selected bay. The Vehicles Awaiting Service list remains unchanged.

Alternate Flows:

[E1]. If no service bay is selected or the selection does not correspond to an actual service bay (index is incorrect), then no vehicle is released and the Service Bay list remains unchanged.

Use Case 9: Quit

Precondition: **The main GUI window is open.

Main Flow: _The user clicks _Quit. The program stops execution.

Design

Trying to jump from requirements right into coding without doing some design first is bound to lead to trouble. We will eventually make you implement our design, but first you need to propose your own. To do this, we give you a scenario and insist on some restrictions for your design.

What you must do: Design Rationale and UML diagram

You must design an application that satisfies the requirements of the Issue Tracker application. You will create a design proposal that shows the objects, their states and behaviors, and the relationships among the objects needed to implement the requirements. Your design must be described in document containing a design rationale and a UML class diagram.

Your design should:

- utilize the Model-View-Controller (MVC) design pattern (see the note about MVC, below).
- contain the [Manageable](#) interface, which we are giving to you.
- contain a GUI (no need to show all its members).

contain at least one additional interface, other than *Manageable*. or at least one abstract class.

contain at least two custom Exception classes.

not rely on any of the Java Collection Frameworks collection classes. (You cannot use the Java-provided List, ArrayList, or LinkedList libraries.)

Answer the following technical questions in your design document as part of the rationale:

1. What objects are important to the system's implementation and how do you know they are important?
2. What data are required to implement the system and how do you know these data are needed?
3. Are the responsibilities assigned to an object appropriate for that object's abstraction and why?
4. What are the relationships between objects (such as inheritance and composition) and why are those relationships are important?
5. What are the limitations of your design? What are the constraints of your system?

MVC Note

Java Swing, the user interface (UI) libraries for Java, does not follow the strictly traditional definition of MVC. Instead, Java Swing utilizes what might be called a [separable model architecture](#). This means that the model is separate and distinct from the view/controller classes that make up the UI. Your design must focus on the model. In your UML class diagram, you should represent the UI as a class with no state or behavior. Your diagram should also show which class(es) your UI will interact with through some type of composition/aggregation/association relationship. The relationship between your model and the UI **must** be justified in your design rationale.

When thinking about the relationships between your UI and the model, consider the following questions:

1. What are the data and behaviors of your model that will be shown through the UI?
2. How does your UI get those data to display; what methods of the model must be called?

Design Proposal and Rationale Submission Format

Submit your design proposal on Moodle under the assignment named Project 2 Part 1 Submission.

Use the [provided design proposal template](#) as a starting point for your design proposal. Use a UML diagramming tool (options are listed the [Software Development Practices notes](#)) to create your UML diagram. Incorporate your UML diagram into your written proposal (using an editing tool such as MS Word) and save the entire document as a PDF. Alternatively, create a PDF for the design proposal and another PDF for the UML diagram, then append the diagram to the proposal to make a *single* PDF document.

Need a little more direction?

See the following example design proposal: [Sample Design Proposal](#).

Testing

This project requires you to do white box and black box testing. You can defer white box testing until Part 2 of this project. But now, you need to prepare some black box test cases.

Scenario

You must create a black box test plan that describes five test cases that will determine if the finished program is satisfies the requirements. Write the tests before you begin development to clarify the inputs to and outputs from the system. Each test must demonstrate that the program satisfies a scenario from the requirements. A scenario is one major path of functionality as described by the requirements.

Assignment

You must write at least five (5) tests for the Wally's Service Garage project. Use the [provided black box test plan template](#) to describe your tests. Each test must be repeatable and specific. All input and expected results values must be concrete. Note that *all* inputs required to complete the test must be specified, including vehicle information files (such as [cars.txt](#)) as well as user input. We highly recommend that you create smaller files with a smaller list of vehicle information for testing.

Additionally, include instructions in your black box test plan for how a tester would set up, start, and run the application for testing (what class from your design contains the main method that starts your program? What are the command line arguments, if any? What do the input files contain?). Describe the instructions at a level where anyone using Eclipse could run your tests.

Format

Use the [provided template](#) as a starting point for your black box test plan. Save your BBTP as a PDF. Submit the PDF to Moodle under the assignment named Project 2 Part 1 Submission.

Need a little help?

See the following example black box test plan: [Sample Black Box Test Plan](#).

Deployment

For this class, deployment means submitting your work for grading. For Part 1 of this programming assignment, you must submit two PDF documents:

1. Design document with incorporated UML class diagram.
2. Black box test plan document.

Make sure that your submissions satisfy the [grading rubrics](#).

You should submit the documents for Project 2 Part 1 to Moodle.

Submission Reminders

The electronic submission deadline is precise. Do not be late. You should count on last minute system failures (your failures, ISP failures, or Gradescope failures). You are able to make multiple submissions of the same file. (Later submissions to a Moodle assignment overwrite the earlier ones.) To be on the safe side, start submitting your work as soon as you have completed a substantial portion.

[Project 2, Part 2: Wolf Travel Tours](#) ►

Published with [GitHub Pages](#)