Riko Hamblin

Train Management System

Problem Description

In this project, a train management system was created. This application keeps track of how many trains there are, where they are going, and also takes care of the capacity using a counter. It allows users to choose what they would like to use the system as. They can choose between being the manager, driver or the passenger. If the capacity of the train reaches its maximum the passenger trying to get in will be informed and will have to select another train. The system also takes care of the position of the train. This means that the system can find out whether the train has reached its destination, is enroute, etc.This functionality was implemented using State design pattern.

The driver assigns itself to a train that already exists and is able to drive it from point A to B via the various States of a train. The manager's job is to create/remove trains and train drivers as needed. The passengers have the option to pick between different trains that exist and once they get on they can get off at the destination that particular train is going to.

Design

The State design pattern was chosen because this type abstraction allows a train to be represented using various States. In the future, as more States are needed to described what the train is capable of doing, the task of adding additional states becomes easy. In reference to our UML diagram, it can be seen that currently there are four states of a Train at this moment (see the concrete implementations of the *State* class) - Enroute, Delayed, AtPickUpLocation, AtDestination. *A change of state results in the logging of that train's state change to a file.*

The Singleton design pattern was also applied to the abstraction of the application that represents the data management system (TrainManagementSystem class). This was chosen because logically it makes sense (and would lead to proper system design) to have only one instance of this abstraction present.

Functional Requirements

Below are three use cases that can occur (See appendix for Use Case Diagram):

1) **Name:** Board Train as passenger
   Participants: Passenger
   Entry Conditions: The user must log in with a valid username
   Flow of events:
   1. The user enters username
   2. They Choose the train they are boarding.
   3. The system then checks the status of the train.
   4. The system checks the capacity of the train to ensure there is space. If there is space the user boards the train.

   Quality Requests:

2) **Name:** Board Train as driver
   Participants: Driver and Customer
   Entry Conditions: The user must log in with a valid username and password. The user must either be a driver.
   Flow of events:
   1. The user enters username and password that match with a driver's. If not then they are not allowed entry
   2. If the driver is already driving a train then they are not allowed to board and can only change the status of their train.
   3. If the driver is not currently attached to a train then they choose one which doesn't have a driver and they are now driving that one.

   Quality Requests:

3) **Name:** Add a Train Driver to system
   Participant: Manager
   Entry conditions: The user must be a manager.
   Flow of events:
   1. The user enters username and password and must be a manager.
   2. Depending on the situation of whether a new train is needed or not, the manager has the option to add a train.
   3. If the manager decides to add a train, they enter a start and end location as well as a capacity. One completely new train is now available.

   Quality Requests:

Testing

Test case ID	Test Case Description	Whitebox/Blackbox
test1	This testing which a value can be successfully found in a file using the FileLogger class	Blackbox
test2	Testing whether a file can be successfully modified	Blackbox
test3	Test whether a line can be successfully removed from a file	Blackbox
test4	Testing boundary condition of Manager class constructor (having a null reference). Ensuring it results in a throw exception	Whitebox
test5	Testing aliasing conditions of Manager class constructor (i.e. when two references point to one object)	Whitebox
test6	Testing boarding Passenger on a train when it is filled	Blackbox
test7	Testing helper method of State class which generates a state based on input	Whitebox
test8	Testing changing state of a train	Blackbox
test9	Testing getCapactiy of a train	Blackbox
test10	Testing getCurrentPassengersAbroad	Blackbox

Five Classes for which specification was provided:

User.java
Passenger.java
TrainDriver.java
Manager.java
Train.java

Conclusion

The objective of this project was to create a software application based on the object oriented techniques that were taught throughout the course. After deciding what the project was going to be about, the next step was to create the UML diagram. The UML use case diagram worked as an outline for the system. It helps the programmer easily modify the structure of the program. The biggest challenge we faced in this project was keeping up with the code. When the code started getting lengthy it became very hard to keep track of the variables and the methods used throughout the code. This is when the UML case diagrams really came in handy. Looking at the class diagram it was easy to analyze all the different components of the code.

The only way to keep the code flowing and understanding relations between different methods was through the use of sequential diagram. The sequential diagram made it very simple to track down the flow of information between methods with related variables. The variables were being passed around from method to method and without the use of the sequential diagram it would have been very hard to keep track of them.

Appendix

Diagrams

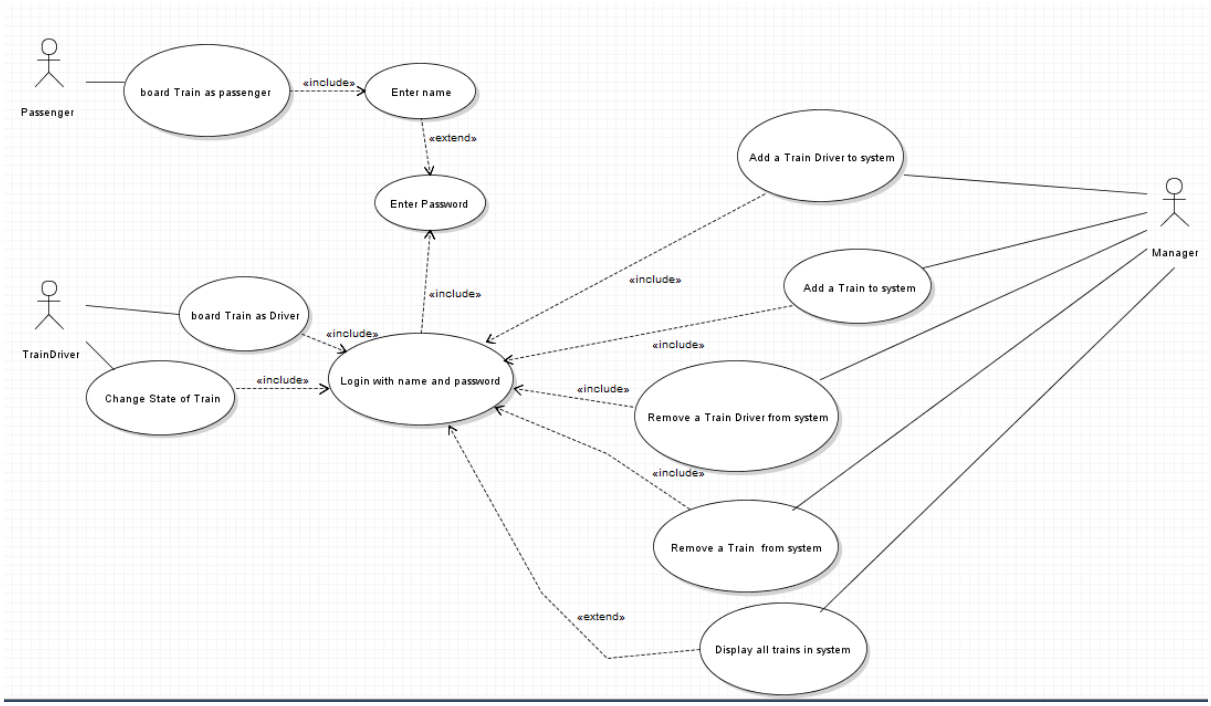

Figure 1. Use Case Diagram

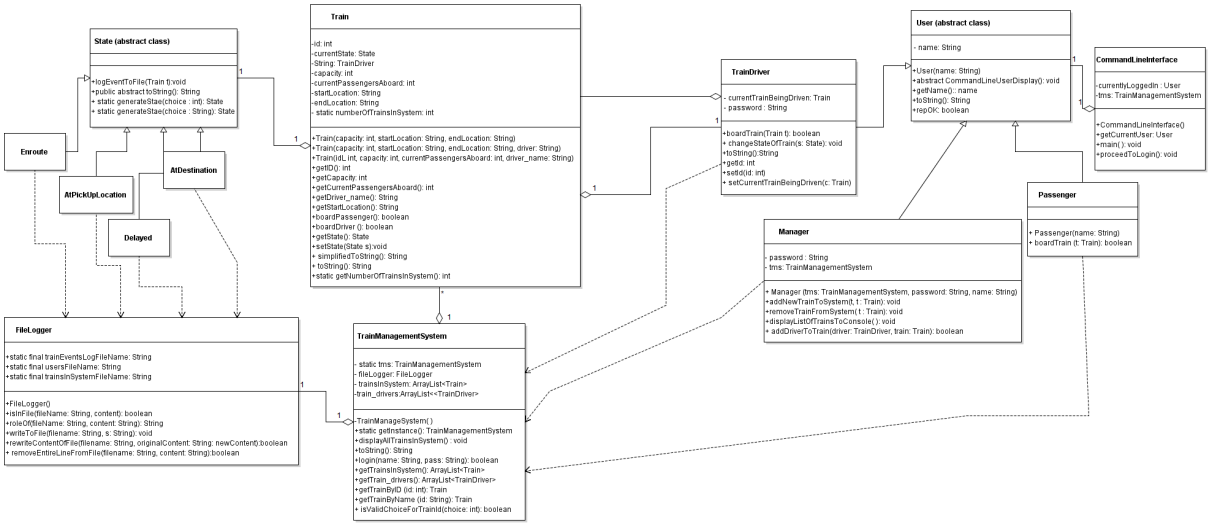

Figure 2a show the class (UML) design of this system.

Due to the size and difficulty to read individual lines the following diagrams were provided. As seen below the magnification will occur first for the red section (See Figure 2.c) then with the blue (See Figure 2.d):

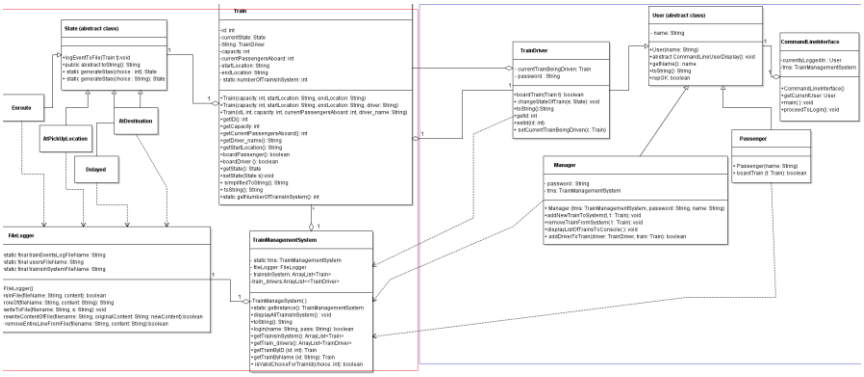

Figure 2b above shows how diagram will be broken down (red square and blue square)

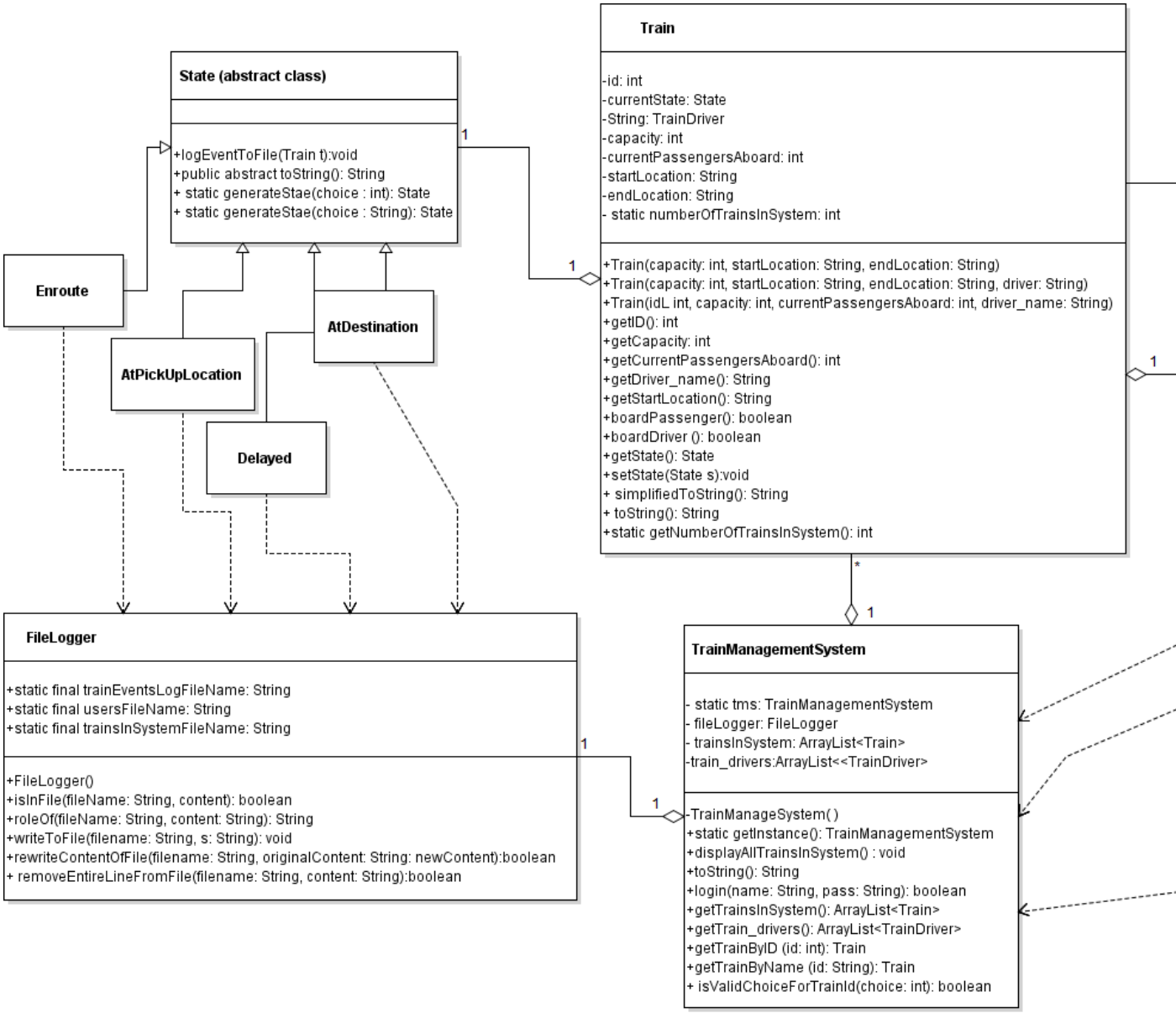

Figure 2c shows half of the system

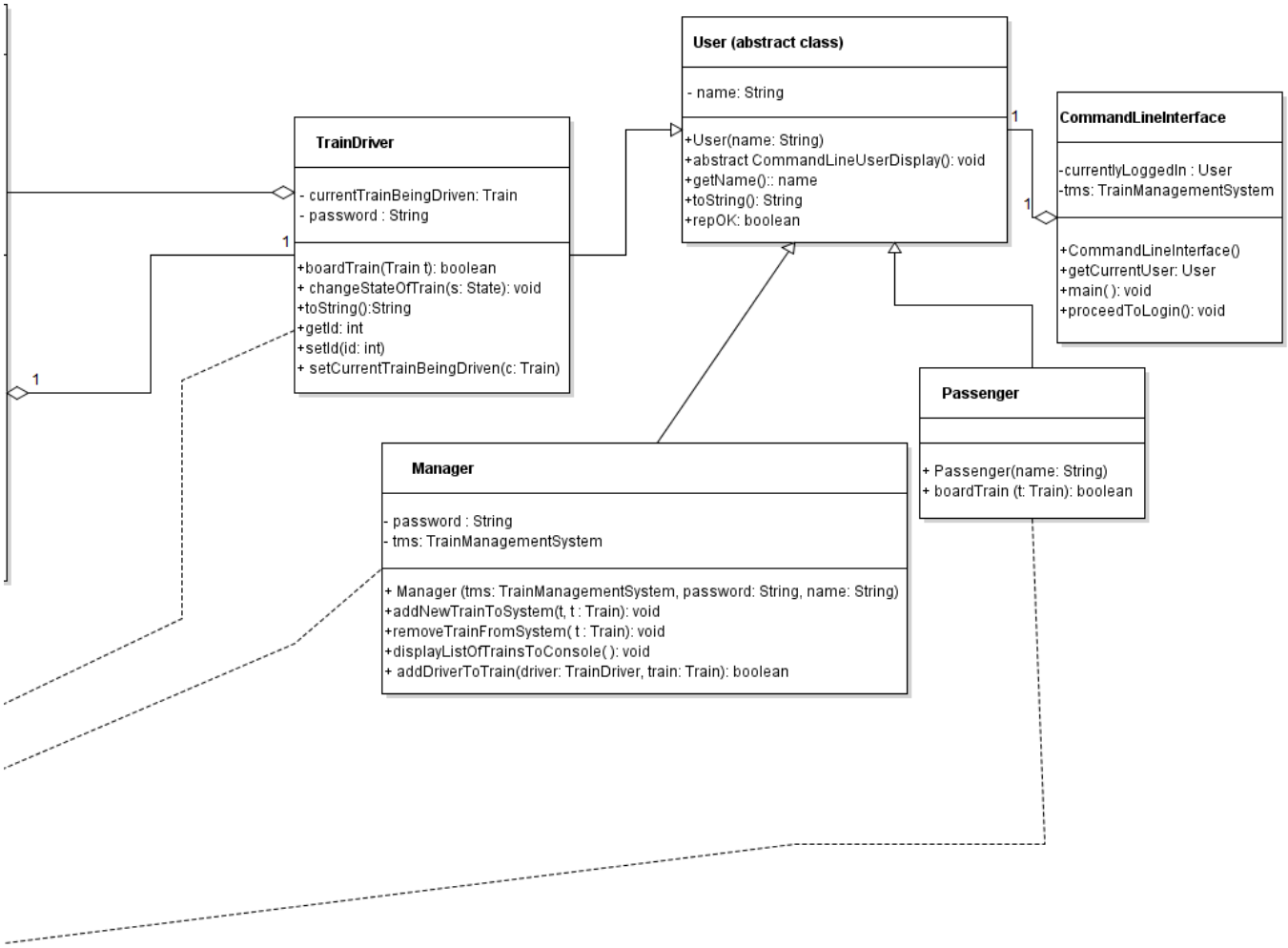

Figure 2d shows the remaining half of the system