PackAir Security Management Design Proposal

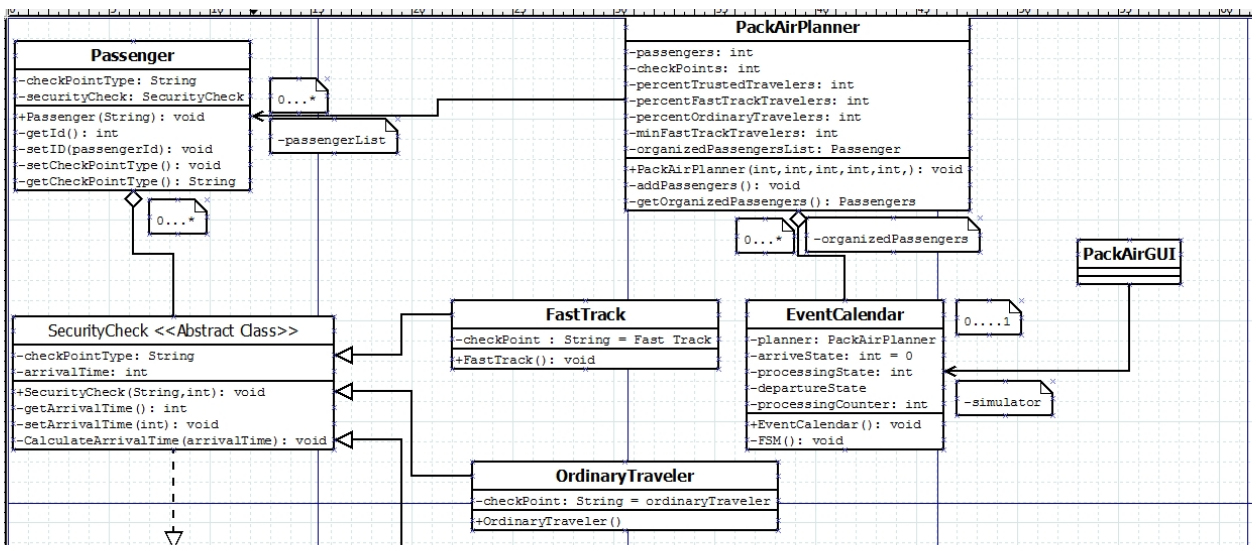
Document Author(s): Richard Howe

Date:10/5/2018

Design Rationale

For the PackAir Security Management program I decided on the use of 9 classes to implement the required behavior. The classes used are *Passenger, PackAirPlanner, SecurityCheck, FastTrack, OrdinaryTraveler, TrustedTSATraveler EventCalendar, PackAirGUI*, and the interface *Calculate*. Listed below is a picture displaying the classes and the relationships between them in my diagram. For my design I used the Model - View – Controller pattern where the PackAirGUI class acts as my view, Passenger, PackAirPlanner, SecurityCheck, and its child classes act as the model, and the Event calendar acts as the controller. The PackAirGUI class starts my program and is also responsible for receiving the user input and displaying the output of the EventCalendar class. PackAirGUI also controls the color coding for each type of security checkpoint simulation.

The EventCalendar class calls the PackAirPlanner to receive an array list of passengers and their chosen checkpoints. The passengers will be sorted in the ArrayList based on their arrivalTime, which will be assigned using the Calculate interface. The event calendar then runs the FSM method which contains a finite state machine (FSM) made up of switch and case statements that simulates passengers going through the security checkpoints. The initial state in the FSM is the arrival state, the next state is the processing state. During this state the variable processCounter will equal the length of the array list, and the processing state will not be finished until processState equals processCounter. The final state is the departure state, which simulates the passengers leaving the checkpoint area and ends the entire simulation.


Passenger
-checkPointType: String
-securityCheck: SecurityCheck
+Passenger(String): void
-getId(): int
-setID(passengerId): void
-setCheckPointType(): void
-getCheckPointType(): String
0...*
-passengerList
0...*
PackAirPlanner
-passengers: int
-checkPoints: int
-percentTrustedTravelers: int
-percentFastTrackTravelers: int
-percentOrdinaryTravelers: int
-minFastTrackTravelers: int
-organizedPassengersList: Passenger
+PackAirPlanner(int,int,int,int,int,): void
-addPassengers(): void
-getOrganizedPassengers(): Passengers
0...*
-organizedPassengers
PackAirGUI
SecurityCheck <<Abstract Class>>
-checkPointType: String
-arrivalTime: int
+SecurityCheck(String,int): void
-getArrivalTime(): int
-setArrivalTime(int): void
-CalculateArrivalTime(arrivalTime): void
FastTrack
-checkPoint : String = Fast Track
+FastTrack(): void
EventCalendar
-planner: PackAirPlanner
-arriveState: int = 0
-processingState: int
-departureState
-processingCounter: int
+EventCalendar(): void
-FSM(): void
0....1
-simulator
OrdinaryTraveler
-checkPoint: String = ordinaryTraveler
+OrdinaryTraveler()

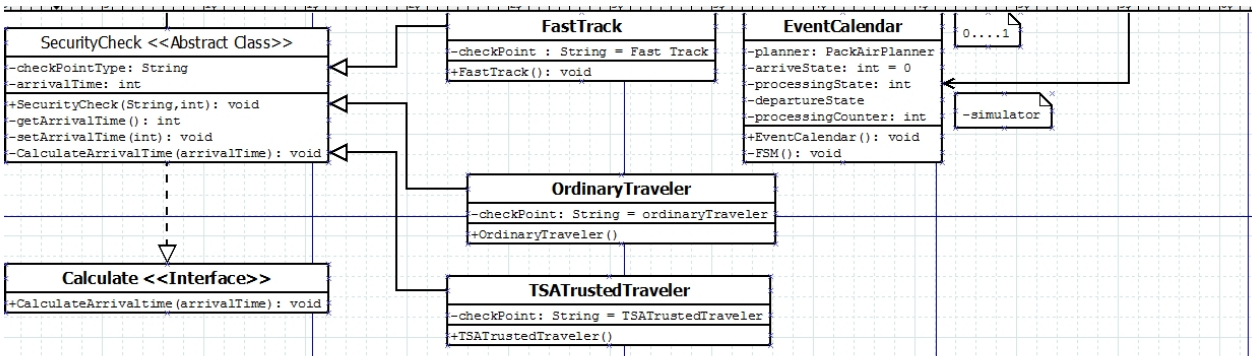


Figure 1: Class Diagram for PackAir Security Management

The PackAirPlanner class receives and validates the user input, and then uses it to create an array list of all created passengers. The Passenger class has an instance of the abstract class SecurityCheck and uses its methods to determine the correct check point for each passenger. The 3 child classes that inherit SecurityCheck contain a string that serves as a label for the chosen check point. They also set the arrival time for their respective passenger. This inheritance relationship allows the child classes to share common methods which helps reduce code redundancy and allows me to group together common classes.

Using the MVC design allows me to incorporate a graphical user interface into my program rather than forcing the customer to rely on an outdated console-based environment. This design is limited due to the fact that the Event Calendars finite state machine can only run through one type of checkpoint at a time. Any implementation of multi-threading would require a redesign as well as verification and inspection by senior project managers.

1) After reading the instructions it looks like the passenger class, its children and the PackAirPlanner class are the most critical to my systems implementation. I know this because after reading the instructions I based these classes off of the directions that were explicitly given in the Problem Overview section of the Project1, Part1:Airport Security page.

2) The inputs into my program are the number of security checkpoints, the number of passengers, and the percentage distribution of passengers. Without these inputs the program can not start. I know that this information is needed because the instructions clearly state that these values are supposed to be the program inputs. Also, the entire program is built around these values.

3) Yes. The child classes to my abstract class contain one field that states what kind of checkpoint it is. Due to the inheritance hierarchy they also have access to all of the abstract classes methods, if they are defined. The child classes will also be using the set/get arrivalTime() methods listed in the parent abstract class. Since the child classes are a type of checkpoint, and the parent class is called SecurityCheck, which is meant to be a broad term for a non-specific type of checkpoint, it makes sense that the child classes have access to the parent classes methods and state.

4)My abstract class has a Generalization relationship with the child classes which allow the child classes to inherit the state and behavior of the parent class. This is important because the child classes need to be able to set the value for the arrivalTime variable. This type of relationship also allows me to incorporate my interface into my abstract class. My SecurityCheck class has a composite relationship with my Passenger class, which allows

Passenger to have an instance of SecurityCheck. Due to this relationship a passenger now *"has a"* checkpoint. EventCalender and PackAirPlanner also have the same kind of relationship. This allows one class to have direct access to all of the other classes methods and states.

5) Using the MVC design allows me to incorporate a graphical user interface into my program rather than forcing the customer to rely on an outdated console-based environment.

Document Revision History

Date	Author	Change Description
10/5/2018	**Richard Howe**	• Wrote the design proposal and added the UML diagram