Pilot-related classes and documentation

A. Overview:

We are supporting for Vietnam Airlines and particularly for models: Boeing 787, Airbus A350, Airbus A320 NEO, and Airbus A321. Each type of model has its own standards for pilots. As I searched for information on the Internet, basic standards are listed below for each type of model:

Standards for a captain who pilots a Boeing 787 model. [3], [5]	
Minimum total flight hours	5000
Hours in command	2000
License type	ATPL
Medical Certificate / Health status	Class 1
English proficiency	Level 4
Age	Under 63 years for male or under 58 for females

Standards for a captain who pilots an Airbus A350 model. [4], [5]	
Minimum total flight hours	3500
Hours in command	1500
License type	ATPL
Medical Certificate / Health status	Class 1
English proficiency	Level 4
Age	Under 63 years for male or under 58 for females

Standards for a captain who pilots an Airbus A320 NEO model. [6]	
Minimum total flight hours	3500
Hours in command	1500
License type	ATPL
Medical Certificate / Health status	Class 1
English proficiency	Level 4
Age	Under 63 years for male or under 58 for females

Standards for a captain who pilots an Airbus A321 model. [7]

Minimum total flight hours	3500
Hours in command	1500
License type	ATPL
Medical Certificate / Health status	Class 1
English proficiency	Level 4
Age	Under 63 years for male or under 58 for females

B. Pilot-related classes and description:

1. The PilotStandard Class:

Purpose: The class holds data about the standards of the pilot or a specific model, which includes the model name, the pilot's minimum flight hours, the pilot's minimum hours in command, the pilot's English level, the pilot's health status, the pilot's license type, and the pilot's maximum ages.

The UML of the class is shown below:

PilotStandard Class
– modelName : string – minRequiredFlightHours : int – minRequiredHoursInCommand : int – requiredLicenseType : string – minRequiredEnglishLevel : int – requiredHealthStatus : int
+ PilotStandard() : + PilotStandard(model : string, flightHours : int, hrsInCommand : int, type : string, english : int, health : int) : + getModelName() : string + getMinRequiredFlightHours() : int + getMinRequiredHoursInCommand() : int + getRequiredLicenseType() : string + getMinRequiredEnglishLevel() : int + getRequiredHealthStatus() : int + setModelName(model : string) : void + setMinRequiredFlightHours(hours : int) : void + setMinRequiredHoursInCommand(hours : int) : void

+ setRequiredLicenseType(type : string) : void
+ setMinRequiredEnglishLevel(level : int) : void
+ setRequiredHealthStatus(status : int) : void
+ operator << (strm : ostream &, standards : PilotStandards) : ostream &

Description of the PilotStandard class:

Members	**Access mode**	**Description**
modelName	Private	The model of aircraft for which the standards are applied for.
minRequiredFlightHours	Private	The minimum required flight hours for the pilot of the model.
minRequiredHoursInCommand	Private	The minimum required hours in command for the pilot of the model.
requiredTypeLicense	Private	The required type of license of the pilot who wants to drives the model.
minRequiredEnglishLevel	Private	The minimum English level of the pilot who wants to drives the model.
requiredHealthStatus	Private	The required health status of the pilot who wants to drives the model.
PilotStandard	Public	The default constructor assigns • An empty string to **modelName** and **requiredLicenseType** • 0 to **minRequiredFlightHours**, **minHoursInCommand**, **minRequiredEnglishLevel**, and **requiredHealthStatus.**
PilotStandard	Public	The constructor accepts 6 arguments for the 6 member variables. It then calls the mutator functions to assign argument to the member variables. The function rethrows exception, when it catches any exceptions.
getModelName	Public	Returns the model name for which the standards of pilot is applied for.

getMinRequiredFlightHours	Public	Returns the minimum required flight hours for the pilot who wants to drive the model.
getMinHoursInCommand	Public	Returns the minimum required hours in command for the pilot who wants to drive the model.
getRequiredLicenseType	Public	Returns the required license type for the pilot who wants to drive the model.
getMinRequiredEnglishLevel	Public	Returns the minimum required English level the pilot must obtain to drive the model.
getRequiredHealthStatus	Public	Returns the required health status that the pilot must obtain to drive the model.
setModelName	Public	The function accepts an argument about the model. It first removes whitespaces and then capitalizes the argument. Next, it checks if the model is a valid model name. If it is invalid, then the function throw an InvalidModel object as an exception. Otherwise, the model is stored in **modelName**.
setMinRequiredFlightHours	Public	The function accepts an argument about the minimum required flight hours. It first checks if the argument is in the valid range. If it is invalid, then the function throw an InvalidHours object as an exception. Otherwise, the argument is stored in **minRequiredFlightHours**.
setMinRequiredHoursInCommand	Public	The function accepts an argument about the minimum required hours in command. It first checks if the argument is in the valid range. If it is invalid, then the function throw an InvalidHours object as an exception. Otherwise, the argument is stored in **minRequiredHoursInCommand**.
setRequiredLicenseType	Public	The function accepts an argument about the required license type. It first removes whitespaces and then capitalizes the argument. Next, it checks if the argument is a valid license type. If it is invalid, then the function throw an InvalidType object as an exception. Otherwise, the model is stored in **requiredLicenseType**.

setMinRequiredEnglishLevel	Public	The function accepts an argument about the minimum required English level. It first checks if the argument is in the valid range. If it is invalid, then the function throw an InvalidEnglish object as an exception. Otherwise, the argument is stored in **minRequiredEnglishLevel**.
setRequiredHealthStatus	Public	The function accepts an argument about the required health status. It first checks if the argument is in the valid range. If it is invalid, then the function throw an InvalidHealth object as an exception. Otherwise, the argument is stored in **requiredHealthStatus**.
operator <<	Public	The overloaded operator << is desgined to display all standards to the screen.

2. The PilotCompetence Class:

The class holds data about the competence of the pilot, which includes the pilot's total flight hours, the pilot's hours in command, the pilot's English proficiency, and the pilot's health status.

The class also contains classes, including InvalidHours, InvalidEnglish, and InvalidHealth for throwing exception when the input is invalid.

The UML of classes for throwing exception:

InvalidHours Class
– value : int
+ InvalidHours(h : int) : + getValue() : int

InvalidEnglish Class
– value : int
+ InvalidEnglish(l : int) : + getValue() : int

InvalidHealth Class
– value : int
+ InvalidHealth(h : int) : + getValue() : int

The UML of the class is shown below:

PilotCompetence Class
– flightHours : int – hoursInCommand : int – englishLevel : int – healthStatus : int
+ PilotCompetence() : + PilotCompetence(hour : int, command : int, english : int, health : int) : + setFlightHours(hours : int) : void + setHoursInCommand(hours : int) : void + setEnglishLevel(newLevel : int) : void + setHealthStatus(newStatus : int) : void + getFlightHours() : int + getHoursInCommand() : int + getEnglishLevel() : int + getHealthStatus() : int + operator << (strm : ostream &, obj : PilotCompetence) : ostream & + operator >> (strm : istream &, obj : PilotCompetence) : istream &

The description of members in the PilotCompetence class:

Members	**Access mode**	**Description**
flightHours	Private	The member variable holds the total flight hours of the pilot.
hoursInCommand	Private	The member variable holds the total number of hours in command of the pilot.
englishLevel	Private	The member variable holds the English proficiency of the pilot. There are a total of six levels from 1 to 6. 1. Level 1: Pre-Elementary 2. Level 2: Elementary 3. Level 3: Pre-operational 4. Level 4: Operational (minimum required for pilots) 5. Level 5: Extended 6. Level 6: Expert

healthStatus	Private	Represents the pilot's **medical certification level**: • Class 1: Highest medical standard, required for commercial pilots. • Class 2: Standard for private pilots. • Class 3: May apply to recreational pilots or air traffic controllers (varies by country). This attribute is used to **evaluate a pilot's medical eligibility** during takeoff inspections or competence comparisons.
PilotCompetence	Public	The default constructor assigns default values to all member variables: • Assigns 0 to the total flight hours. • Assigns 0 to the number of hours in command. • Assigns 0 to the English proficiency. • Assigns 0 to the status of health
PilotCompetence	Public	The constructor takes arguments and stores it in corresponding member variables. It calls the accessor functions to assigns data to member variables. Accessor functions throw exception, so the constructor will rethrow the exception.
setFlightHours	Public	The member function takes an argument and stores it in the *flightHours* member variable. The function also performs validation of the argument passed into the function. The argument should be non-negative. If the argument is negative, then the function throws an InvalidHours object with invalid data as an exception.
setHoursInCommand	Public	The member function takes an argument and stores it in the *hoursInCommand* member variable. The function also performs validation of the argument passed into the function. The argument should be non-negative. If the argument is negative, then the function throws an InvalidHours object with invalid data as an exception.
setEnglishLevel	Public	The member function takes an argument and stores it in the *englishLevel* member variable. If the argument is out of the valid range, then the function throws an InvalidEnglish object with invalid data as an exception.

setHealthStatus	Public	The member function takes an argument and stores it in the *healthStatus* member variable. If the argument is out of valid range, then the function throws an InvalidHealth object with invalid data as an exception.
getFlightHours	Public	The member function returns the value of the *flightHours* member variable.
getHoursInCommand	Public	The member function returns the value of the *experienceYears* member variable.
getEnglishProficiency	Public	The member function returns the value of the *englishProficiency* member variable.
getHealthStatus	Public	The member function returns the value of the *healthStatus* member variable.
compareWithStandard	Public	The member function compares the pilot with the standard and writes the details of the inspection result.
operator <<	Public	The overloaded << operator is used to print the details of a PilotCompetence object. It is going to output the pilot's flight hours, experience years, English proficiency level, and health status.
operator >>	Public	The overloaded >> operator is used to get the input of a PilotCompetence object. It is going to take input of the pilot's flight hours, experience years, English proficiency level, and health status.

3. The PilotCertificate class:

The PilotCertificate class holds official information about the pilot's license including the license type, the license number, and the expiry data of the license. The class also provides methods to store and retrieve the member variables of the class and methods to inspect the validity of the pilot's license.

The class contains a class for throwing exception, which is InvalidType, when the license type is invalid. The UML of the InvalidType class is shown below:

InvalidType Class
– type : string
+ InvalidType(t : string) : + getType() : string

The UML of the class is shown below:

PilotCertificate Class
– licenseType : string – expiryDate : Date
+ PilotCertificate() : + PilotCertificate(type : string, date : Date) : + setLicenseType(newType : string) : void + setExpiryDate(newDate : string) : void + getLicenseType() : string + getExpiryDate() : Date + isLicenseExpired() : bool + operator << (strm : ostream &, obj : PilotCertificate) : ostream & + operator >> (strm : istream &, obj : PilotCertificate) : istream &

Description of members in the PilotCertificate class:

Members	**Access mode**	**Description**
licenseType	Private	The member variable holds the type of the license of the pilot. There are the following types of licenses for pilots: • **SPL (Sport Pilot License):** permits individuals to fly a light-sport aircraft (LSA) at low altitudes in their local area. Those with this certification can fly with one passenger. There are limits, including day flying in areas below 10,000 feet.[1] • **RPL (Recreational Pilot License):** allows an individual to fly slightly heavier aircraft with up to 190 horsepower, up to 50 nautical miles from their departure airport. It's limited to day-flying with up to one passenger in non-controlled airspace.[1] • **PPL (Private Pilot License):** Allows the holder to fly aircraft for personal, non-commercial purposes. With this license, a pilot can carry passengers, fly at night, and travel long distances, even in different countries—as long as it’s not for pay.[2] • **CPL (Commercial Pilot License):** allows a person to get paid to fly. With this license, a pilot can work

		as a professional pilot, for example, flying cargo, doing aerial surveys, or working as a co-pilot for an airline.[1] • **ATPL (Airline Transport Pilot License):** authorizes a pilot to fly for a major airline, required to captain airline flights; highest level with the most experience.[1]
expiryDate	Private	The member variable holds the expiration date of the pilot's license in. It is a Date object
PilotCertificate	Public	The default constructor assigns default values to member variables: • Assigns an empty string to the license type. • Assigns an empty string to the license number. • Assigns default data to the expiry date
PilotCertificate	Public	The constructor accepts arguments and stores them in corresponding member variables. It calls accessor functions to assign arguments to member variables, and if any exception is caught, then the constructor rethrows the exception.
setLicenseType	Public	The member function accepts an argument and stores it in the *licenseType* variable. The function also performs the validation of the argument to determine if the argument is valid. If the argument is invalid, then the function throws an InvalidType object with invalid data as an exception.
setLicenseNumber	Public	The member function accepts an argument and stores it in the *licenseNumber* variable.
setExpiryDate	Public	The member function accepts an argument and stores it in the *expiryDate* variable.
getLicenseType	Public	The member function returns the license type of the pilot.
getLicenseNumber	Public	The member function returns the license number of the pilot.
getExpiryDate	Public	The member function returns the expiration date of the pilot’s license.
isLicenseExpired	Public	The member function checks whether the pilot’s license is out of date. If the license is expired, then the function returns True. Otherwise, it returns False.

operator <<	Public	The overloaded operator << is designed to print details of the pilot's certificate out.
operator >>	Public	The overloaded operator >> is designed to get input of a PilotCertificate object including license type, license number, and expiry date.

4. The Pilot Class:

The Pilot class holds information about a pilot including name, age, competence, certificate. The class also provides methods to store and retrieve member variables.

The class also contains classes, including InvalidName, InvalidAge, and InvalidGender, for throwing exception, when the input data is invalid. The UML of the three sub-classes are shown below:

InvalidName Class
– value : string
+ InvalidName(n : string) : + getValue() : string

The UML of the Pilot class is shown below:

Pilot Class
– name : string – pilotCompetence : PilotCompetence – pilotCertificate : PilotCertificate
+ Pilot() : + Pilot(pilotName : string, pilotCompetence : PilotCompetence, pilotCertificate : PilotCertificate) : + setName(newName : string) : void + setPilotCompetence(newCompetence : PilotCompetence) : void + setPilotCertificate(newCertificate : PilotCertificate) : void + getName() : string + getPilotCompetence() : PilotCompetence + getPilotCertificate() : PilotCertificate + operator << (strm : ostream &, obj : Pilot &) : ostream & + operator >> (strm : istream &, obj : Pilot &) : istream &

Description of members in the Pilot class:

Members	Access mode	Description
name	Private	The member variable holds the name of the pilot.
pilotCompetence	Private	The member variable holds information about the competence of a pilot, which is a PilotCompetence object (aggregation). The variable consists of total flight hours, hours in command, English proficiency, and health status.
pilotCertificate	Private	The member variable holds information about the official license of a pilot, which is a PilotCertificate object (aggregation). The variable consists of license type, license number, and expiry date of the license.
Pilot	Public	The default constructor assigns default values for all member variables: • Assigns an empty string to **name**. • Assigns default values to **pilotCompetence** (by the default constructor of the PilotCompetence class). • Assigns default values to **pilotCertificate** (by the default constructor of the PilotCertificate class).
Pilot	Public	The constructor accepts arguments and stores them in corresponding member variables. The constructor calls mutator functions to assign arguments to member variables and, if any exception is caught, it rethrows the exception.
setName	Public	The member function accepts an argument and stores it in the **name** variable. The function checks if the name contains invalid characters. If the name is invalid, the function throws an InvalidName object with the invalid name value as an exception.
setPilotCompetence	Public	The member function accepts an argument and stores it in the **pilotCompetence** variable.
getPilotCertificate	Public	The member function accepts an argument and stores it in the **pilotCertificate** variable.
getName	Public	The member function returns the name of the pilot.
getPilotCompetence	Public	The member function returns the competence of the pilot.
getPilotCertificate	Public	The member function returns the certificate about license of the pilot.

operator <<	Public	The overloaded operator << is designed to print details of the pilot's information out.
operator >>	Public	The overloaded operator >> is designed to get input of a Pilot object including name, age, competence, and certificate.

5. Date class:

The class holds the month, day, and year of a date. The function is aggregated in the PilotCertificate class to represent the expiration date of the pilot's license.
The UML of the class is shown below:

Date Class
– month : int – day : int – year : int
+ Date() : + Date(m : int, d : day, y : int) : + getMonth() : int + getDay() : int + getYear() : int + getDate() : string + setMonth(newMonth : int) : void + setDay(newDay : int) : void + setYear(newYear : int) : void + isLeapYear() : bool + operator > (date : Date) : bool + operator >> (strm : istream &, obj : Date) : istream & + opeartor << (strm : ostream &, obj : Date) : ostream &

Description of member variables and member functions of the Date class:

Members	**Access mode**	**Description**
month	Private	The member variable to hold the month of the date.
day	Private	The member variable to hold the day of the date.
year	Private	The member variable to hold the year of the date.
Date	Public	The default constructor assigns 0 to all three member variables.

Date	Public	The constructor accepts three arguments about the month, day, and year of a date and calls mutator functions to assigns the arguments to member variables. The constructor rethrows exceptions if it catches any exceptions.
setYear	Public	The member function accepts an argument and stores it in the **year** variable. The function checks if the argument is negative or not. If the argument is negative, then the function throws an InvalidYear object as an exception. **InvalidYear** – value : int + InvalidYear(y : int) : + getValue() : int
setMonth	Public	The member function accepts an argument and stores it in the **month** variable. The function checks if the argument is valid (from 1 to 12). If the argument is invalid, then the function throws an InvalidMonth object as an exception. **InvalidMonth** – value : int + InvalidMonth(m : int) : + getValue() : int
setDay	Public	The member function accepts an argument and stores it in the **day** variable. The function checks if the argument is a valid day for the month. If the argument is invalid, then the function throws an InvalidDay object as an exception. **InvalidDay** – value : int + InvalidDay(d : int) : + getValue() : int
getYear	Public	The member function returns the value in the **year** member variable.
getMonth	Public	The member function returns the value in the **month** member variable.
getDay	Public	The member function returns the value in the **day** member variable.
getDate	Public	The member function returns a string of the date in the format of MM/DD/YYYY.

isLeapYear	Public	The member function returns True if the **year** is a leap year. Otherwise, it returns False.
operator <<	Public	The overloaded operator will display the date in a format of MM/DD/YYYY
operator >>	Public	The overloaded operator will get data for a Date object.

6. InpsectionResult class (Abstract base class):

The class represents a paper to hold the result of inspection. It contains the title, the notes, and the inspection result. Here is the UML of the class.

InpsectionResult Class
title : string # inspectionResult : bool # notes : vector<string> # setInspectionResult() : void = 0
+ InspectionResult() : + InspectionResult(t : string) : + getTitle() : string + getInspectionResult() : bool + getNotes() : vector<string> + addNotes(newNote : string) : void + displayNotes() : void + setTitle(newTitle : string) : void {virtual} + ~InspectionResult() :

Description of the member variables and member functions of the InspectionResult class:

Members	**Access mode**	**Description**
title	Private	The title of the inspection result.
inspectionResult	Private	The overall inspection result: True (= Eligible) and False (= Ineligible)
notes	Private	The vector to holds notes of the inspection result.
InspectionResult	Public	The default constructor assigns an empty string to **title**, assigns False to **inspectionResult.**

InspectionResult	Public	The constructor accepts an argument about the title of the inspection result and assigns it to **title**, and assigns False to **inspectionResult**.
getTitle	Public	The member function returns the value in **title** member variable.
getInspectionResult	Public	The member function returns the value in **inspectionResult** member variable.
getNotes	Public	The member variable returns the vector of **notes**.
addNotes	Public	The member accepts a string and pushes it back to the **notes** vector.
displayNotes	Public	The member function displays all the notes in the **notes** vector to the screen.
setTitle	Public	The member function accepts and assigns the argument to the **title** member variable.
getDate	Public	The member function returns a string of the date in the format of MM/DD/YYYY.
setInspectionResult	Public	The pure virtual function should be overridden in the derived classes because each aspects have different criteria to inspect.
~InspectionResult	Public	The default virtual destructor will ensure that the destructors of the derived classes to be called when using polymorphism.

7. PilotInspectionResult (Derived from InspectionResult)

The class inherits from the InspectionResult class and has additional member variables to hold the inspection results for the total flight hours, hours in command, license type, English level, health status, and the expiry date of the license. Here is the UML of the class:

PilotInspectionResult Class
flightHoursResult : bool # hoursInCommandResult : bool # englishLevelResult : bool # healthStatusResult : bool # licenseTypeResult : bool # licenseExpiryResult : bool {virtual} # setInspectionResult() : void override

+ PilotInspectionResult() :
+ PilotInspectionResult(t : string) :
+ setFlightHoursResult(result : bool) : void
+ setHoursInCommandResult(result : bool) : void
+ setEnglishLevelResult(result : bool) : void
+ setHealthStatusResult(result : bool) : void
+ setLicenseTypeResult(result : bool) : void
+ setLicenseExpiryResult(result : bool) : void
+ getFlightHoursResult() : bool
+ getHoursInCommandResult() : bool
+ getEnglishLevelResult() : bool
+ getHeathStatusResult() : bool
+ getLicenseTypeResult() : bool
+ getLicenseExpiryResult() : bool
{virtual} + ~PilotInspectionResult() :

Description of member variables and member functions of the PilotInspectionResult class:

Members	**Access mode**	**Description**
flightHoursResult	Private	The result of the pilot's flight hours. True means the pilot's flight hours is met. False means doesn't meet.
hoursInCommandResult	Private	The result of the pilot's hours in command. True means the pilot's hours in command is meet. False means doesn't meet.
englishLevelResult	Private	The result of the pilot's English level. True means the pilot's English level is met. False means doesn't meet.
healthStatusResult	Private	The result of the pilot's health status. True means the pilot's health status is met. False means doesn't meet.
licenseTypeResult	Private	The result of the pilot's license type. True means the pilot's license type is met. False means doesn't meet.
PilotInspectionResult	Public	The default constructor calls the default base class constructor, assigns False to all 6 member variables, and calls the function **setInspectionResult** to update the inspection result.
PilotInspectionResult	Public	The constructor accepts an argument about the title, passes the argument to the base class constructor,

		assigns False to all 6 member variables, and calls the function **setInspectionReult** to update the inspection result.
licenseExpiryResult	Public	The result of the pilot's license expiration. True means the pilot's license has not expired. False means the pilot's license has expired.
setFlightHoursResult	Public	The member function accepts an argument, assigns it to **flightHoursResult**, and calls the **setInspectionResult** function to update the inspection result.
setHoursInCommandResult	Public	The member function accepts an argument, assigns it to **hoursInCommandResult**, and calls the **setInspectionResult** function to update the inspection result.
setEnglishLevelResult	Public	The member function accepts an argument, assigns it to **englishLevelResult**, and calls the **setInspectionResult** function to update the inspection result.
setHealthStatusResult	Public	The member function accepts an argument and assigns it to **healthStatusResult,** and calls the setInspectionResult function to update the inspection result.
setLicenseTypeResult	Public	The member function accepts an argument and assigns it to **licenseTypeResult,** and calls the setInspectionResult function to update the inspection result.
setLicenseExpirtyResult	Public	The member function accepts an argument and assigns it to **licenseExpiryResult**, and calls the setInspectionResult function to update the inspection result.
getFlightHoursResult	Public	The member function returns the value in **flightHoursResult** member variable.
getHoursInCommandResult	Public	The member function returns the value in **hoursInCommand** member variable.
getEnglishLevelResult	Public	The member function returns the value in **englishLevelResult** member variable.
getHealthStatusResult	Public	The member function returns the value in **healthStatusResult** member variable.

getLicenseTypeResult	Public	The member function returns the value in **licenseTypeResult** member variable.
getLicenseExpiryResult	Public	The member function returns the value in **licenseExpiryResult** member variable.
setInspectionResult	Public	The member function returns True if all of the six member variables are True. Otherwise, it returns False.
~PilotInspectionResult	Public	The default virtual destructor will ensure that the destructors of the derived classes to be called when using polymorphism.

8. StringManipulator class:

The class contains no member variables but provides methods to process a string. The UML of the class.

StringManipulator Class
{static} + capitalize(inputStr : string) : string {static} + removeSpaces(inputStr : string) : string

Description of the member variables of the StringManipulator class:

Members	**Access mode**	**Description**
capitalize	Public	The member function accepts a string, capitalizes all the letters in the string, and returns the output string.
removeSpaces	Public	The member function accepts a string, removes all whitespaces in the string, and returns the output string.

9. FlightInspectionDepartment class

The class performs inspection. Here is the UML of the class:

FlightInspectionDepartment Class
{static} + inspectPilot(pilotInfo : Pilot, standard : PilotStandard) : PilotInspectionResult

Description of the member functions in the FlightInspectionDepartment class:

Members	**Access mode**	**Description**
inspectPilot	Public	The member function accepts a Pilot object and a PilotStandard object. Then, it compares the Pilot with the PilotStandard and returns the a PilotInsepctionResult, which contains all the details of the inspection result.

10. FlightManagementDepartment class

The class is responsible for holding pilot standards, loading pilot standards from the file, find the pilot standards for the model, writing the vector of flights to files.

FlightManagementDepartment Class
{static} – pilotStandardArray : vector<PilotStandard>
{static} + loadPilotStandard(fileName : string) : void {static} + findPilotStandard(model : string) : PilotStandard {static} + displayPilotStandards() : void

Description of member functions in the FlightManagementDepartment class

Members	**Access mode**	**Description**
pilotStandardArray	Private	The member variable belongs to the class itself. It holds PilotStandards object.
loadPilotStandard	Public	The member function accepts a file name and loads the Pilot Standards from the file to the **pilotStandardArray** vector.
findPilotStandard	Public	The member variable accepts the name of a model, finds and return the corresponding PilotStandard object.
displayPilotStandards	Public	The member variable displays all the PilotStandard objects in the **pilotStandardArray** vector.

Input validation:

1. Input validation for the Date class

We assume that the user type data in the format MM DD YYYY or M D YYYY.
The value of month, day year should not be negative. The value of month should be from 1 to 12, and the value of day should be valid depending on the month and the year (whether leap year or regular year).
The mutator functions of the class will perform input validation for month, day, and year. If the value of the month is invalid, then the function throw an exception. If the value of the day is invalid, then the function throw an exception. If the value of the year is invalid, then the function throw an exception.

2. Input validation for the PilotCompetence class

The flightHours member variable should not be negative. If it is negative, the mutator function (setFlightHours) will throw an exception.
The hoursInCommand member variable should not be negative and should be less than the flightHours. If the value of hoursInCommand is invalid, the mutator function (setHoursInCommand) will throw an exception.
The englishLevel member variable should be from 1 to 6. If the value of it is invalid, the mutator function (setEnglishLevel) will throw an exception.
The healthStatus member variable should be from 1 to 3. If the value is invalid, the mutator function (setHealthStatus) will throw an exception.

3. Input validation for the PilotCertificate class

The type of license should contain alphabetical characters and whitespace characters only. The mutator function (setLicenseType) performs validation. If the type of license is invalid, then the function will throw an exception.

4. Input validation for the Pilot class

The name of the pilot class should not contain any characters other than whitespace characters and letters. If the name of the pilot is invalid, the mutator function (setName) will throw an exception.

REFERENCES

[1] “7 Types of Pilot Certifications and Licenses” published by Indeed on March 26, 2025.
https://www.indeed.com/career-advice/career-development/pilot-certifications

[2] “Private Pilot License: 4 Steps to Your PPL In 2025” published in Flight Academy.
https://epicflightacademy.com/private-pilot-course/

[3] “Vietnam Airlines B787 Captain” published in RishworthAviation
https://rishworthaviation.com/job/vietnam-airlines-b787-captain?source=google.com

[4] “Vietnam Airlines A350 Captain” published in AviaCV
https://www.aviacv.com/job/vietnam-airlines-a350-captain/2232?utm_source=google.com

[5] “Vietnam Airlines Pilot Careers & Salary: A Comprehensive Guide” published in Flight Academy
https://epicflightacademy.com/hiring-requirements-vietnam-airlines/?utm_source=google.com

[6]
https://pilotsglobal.com/job/A320-family-captain-vietnam_airlines-VN-2023e66036?utm_source=google.com

[7]
https://www.aviacv.com/job/vietnam-airlines-a320-captains-urgent-requirement-screening-dates-to-be-announced-soon-1/10?utm_source=google.com