Pilot Inspection Design

A. <u>Inspection overview:</u>

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

I plan to create a file which holds standards of pilot for each model and then the program reads the file to store the standards in an array in main. After that, the array is passed into the **FlightInspectionDepartment** class to perform inspection. I am going to design five classes: **PilotStandard**, **PilotCompetence**, **PilotCertificate**, **Pilot**, and **InpsectionResult**. Pilot class will have a PilotCompetence and a PilotCertificate. The PilotStandard class will hold data about the standards of the pilot for a specific model and provide a method to read standards from a file and store the standards in a dynamically allocated array in the program. The Pilot class will hold data about a real-life pilot and provide methods to determines if the pilot exceeds or meets the requirements.

The **PilotInspectionResults** class which inherits from the InspectionResult class can only be accessed by the FlightInspectionDepartment. It will hold the inpsection result of the pilot including a variable to indicate if the pilot is eligible or ineligible. If the pilot is ineligible, all the reasons will be stored in the result.

B. Class design and description:

1. The InspectionResult Class:

The class holds the inspection results of a pilot. It will contain a variable to indicate whether the pilot is eligbile or ineligible. If the pilot is ineligible, all of the reasons why the pilot is incapable should be stored in the result object.

The UML of the class is shown below:

InspectionResult Class
– checkedAspect : string
– results : bool
– unmetCriteria : vector<string>
– setResults(status : bool) : void
– addUnmetCriteria(reason : string) : void
– InspectionResults(object : string) :
+ getResults() : string

+ getCheckedAspect() : string + displayInspectionResults() : void

Description of the members in PilotInspectionResults class:

Members	**Access mode**	**Description**
checkedAspect	Private	The member variable holds which aspect of the flight the inspection result belongs to. The values of the variable should be Pilot, Weather, or Plane.
results	Private	The member variable holds the result of the pilot's inspection. It holds a True value only if the pilot meets the standards. Otherwise, it holds a False value.
unmetCriteria	Private	This is a vector of string to hold criteria that the pilot doesn't meet.
InspectionResults()	Private	Accepts an argument and stores it in the *checkedAspect* variable. Assigns default default values for the two member variables: • Assigns False to the results. • Assigns nothing to the vector.
addUnmetCriteria	Private	The member function takes an argument and appends it to the vector of unmet criteria.
getResults	Public	Returns the value stored in *results* member.
getCheckedAspect	Public	Returns the value stored in *checkedAspect* member.
displayInspectionResults	Public	The member function displays the results of the inspection and the reasons (if necessary).

2. The PilotStandard Class:

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.
Because the standards of the pilot should not be expected to be modified by users, mutator functions are going to be private only, which are only used to store standards from a file in the variables.
The UML of the class is shown below:

PilotStandard Class
– modelName : string – minRequiredFlightHours : int – minRequiredHoursInCommand : int – requiredLicenseType : string – minRequiredEnglishLevel : int – requiredHealthStatus : int – maxAgeMale : int – maxAgeFemale : int
+ getModelName() : string + getMinRequiredFlightHours() : int + getMinRequiredHoursInCommand() : int + getRequiredLicenseType() : string + getMinRequiredEnglishLevel() : int + getRequiredHealthStatus() : int + getMaxAgeMale() : int + getMaxAgeFemale() : int + <<friend>> operator << (strm : ostream &, standards : PilotStandards) : ostream & + loadStandardsFromFile(arrPtr : *PilotStandards, inputFile : ifstream &, numOfRecords : int) : bool

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.

minRequiredEnglishProficiency	Private	The minimum English proficiceny of the pilot who wants to drives the model.
requiredHealthStatus	Private	The required health status of the pilot who wants to drives the model.
maxAgeMale	Private	The maximum age of a male pilot who wants to drives the model.
maxAgeFemale	Private	The maximum age of a female pilot who wants to drives the model.
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.
getMaxAgeMale	Public	Returns the maximum age for a male pilot who wants to drive the model.
getMaxAgeFemale	Public	Returns the maximum age for a female pilot who wants to drive the model.
operator <<	Public	The overloaded operator << is desgined to display all standards to the screen.
loadStandardsFromFile	Public	Accepts a pointer to an array of PilotStandard, an inputFile name, and an integer to indicate how many records in the file the program wants to read. The function reads and stores standards in the array using setter functions. If the number of records in the file is less than the number of records we want to read, default information will be assigns for these records.

3. 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	**InvalidEnglish Class**	**InvalidHealth Class**
– value : int	– value : int	– value : int
+ InvalidHours(h : int) : + getValue() : int	+ InvalidEnglish(l : int) : + getValue() : 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(newHour : int) : void + setEnglishLevel(newLevel : int) : void + setHealthStatus(newStatus : int) : void + getFlightHours() : int + getHoursInCommand() : int + getEnglishLevel() : int + getHealthStatus() : int + <<friend>> operator << (strm : ostream &, obj : PilotCompetence) : ostream & + <<friend>> 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.
operator >=	Public	Accepts an PilotStandards object. The overloaded operator compares the current PilotCompetence object with a required competence standards to determine if the pilot meets or exceeds all required qualifications. It returns true only if the pilot's flight hours, hours in command, English level, and health status are all greater than or equal to those of the required competence.
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.

4. 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 – licenseNumber : string – expiryDate : Date
+ PilotCertificate() : + PilotCertificate(type : string, license : string, date : Date) : + setLicenseType(newType : string) : void + setLicenseNumber(newLicense : string) : void + setExpiryDate(newDate : string) : void + getLicenseType() : string + getLicenseNumber() : string + getExpiryDate() : Date + isLicenseExpired() : bool + <<friend>> operator << (strm : ostream &, obj : PilotCertificate) : ostream & + <<friend>> 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]
licenseNumber	Private	The member variable holds the unique identifier for the pilot's license.
expiryDate	Private	The member variable holds the expiration date of the pilot's license in YYYY-MM-DD format.
PilotCertificate	Public	The default constructor assigns default values to member variables:

		• Assigns “blank” to the license type. • Assigns “blank” 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.

5. 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	**InvalidAge Class**	**InvalidGender Class**
– value : string	– value : int	– value : int
+ InvalidName(n : string) :	+ InvalidAge(a : int) :	+ InvalidGender(g : char) :
+ getValue() : string	+ getValue() : int	+ getValue() : char

The UML of the Pilot class is shown below:

Pilot Class
– name : string
– age : int
– gender : char
– pilotCompetence : PilotCompetence
– pilotCertificate : PilotCertificate
+ Pilot() :
+ Pilot(pilotName : string, pilotAge : int, pilotGender : char,
PilotCompetence : PilotCompetence,
pilotCertificate : PilotCertificate) :
+ setName(newName : string) : void
+ setAge(newAge : int) : void
+ setGender(newGender : char) : void
+ setPilotCompetence(newCompetence : PilotCompetence) : void
+ setPilotCertificate(newCertificate : PilotCertificate) : void
+ getName() : string
+ getAge() : int
+ getGender() : char
+ getPilotCompetence() : PilotCompetence
+ getPilotCertificate() : PilotCertificate
+ isAgeValid(maxAge : int) : bool
+ <<friend>> operator << (strm : ostream &,
obj : Pilot &) : ostream &
+ <<friend>> 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.
age	Private	The member variable holds the age of the pilot.
gender	Private	The member variable holds the gender 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 “blank” to name. • Assigns 0 to age. • Assigns ‘ ‘ to gender. • Assigns default values to competence (by the default constructor of the PilotCompetence class). • Assigns default values to certificate (by the default constructor of the PilotCertificate class).
Pilot	Public	The constructor accepts arguments and stores them in corresponding member variables. The constructor calls accessor 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.
setAge	Public	The member function accepts an argument and stores it in the *age* variable. The function also performs the validation of the argument to determine if the argument is valid. If the age is negative, the function throws an InvalidAge object with the invalid age value as an exception.

setGender	Public	The member function accepts an argument and stores it in the *gender* variable. The function also performs the validation of the argument to determine if the argument is valid. If the gender is invalid, the function throws an InvalidGender object with the invalid gener 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.
getAge	Public	The member function returns the age of the pilot.
getGender	Public	The member function returns the gender 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.
isAgeValid	Public	The member function accepts an argument as the maximum age of the pilot and checks if the age of the pilot is valid. If the age of the pilot is met, the function returns True. Otherwise, it returns False.
operator >=	Public	Accepts a PilotStandards object. The function checks if the pilot exceeds or meets the standards. The function returns True only if the pilot exceeds or meets the standards. Otherwise, it returns False.
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.

C. 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

For simplicity, we assume that the license number is correct. The class only validates the type of license.
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.
The age of the pilot should not be negative. If the age of the pilot is invalid, the mutator function (setAge) will throw an exception.
The gender of the pilot is M or m (for male) or F or f (for female). If the gender is invalid, the mutator function (setGender) 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