Group 1: Vietnam Airlines Flight Inspection

# Pilot-related classes and documentation (Ly Ba Hoang - 1695172):

We support flights in Vietnam Airlines and particularly for four 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 |

|  |  |
| --- | --- |
| **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 |

|  |  |
| --- | --- |
| **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 |

|  |  |
| --- | --- |
| **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 |

## 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, and the pilot’s license type. 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 member variables and member functions in 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 is going to fly the model. |
| minRequiredEnglishLevel | Private | The minimum English level of the pilot who is going to fly the model. |
| requiredHealthStatus | Private | The required health status of the pilot who is going to fly 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 arguments to the member variables. The function rethrows exceptions, when it catches any exceptions. |
| getModelName | Public | Returns the model name for which the standards of pilot are 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 is going to fly the model. |
| getMinRequiredEnglishLevel | Public | Returns the minimum required English level the pilot must obtain to fly the model. |
| getRequiredHealthStatus | Public | Returns the required health status that the pilot must obtain to fly 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 throws 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 throws 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 throws 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 throws 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 throws 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 throws an InvalidHealth object as an exception. Otherwise, the argument is stored in **requiredHealthStatus**. |
| operator << | Public | The overloaded operator displays all standards to the screen. |

## 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 exceptions when the input is invalid. 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 member variables and member functions 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 mutator functions to assign arguments to the member variables. Mutator functions throw exceptions, so the constructor will rethrow the exceptions. |
| 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.   |  | | --- | | **InvalidHours Class** | | − value : int | | + InvalidHours(h : int) :  + getValue() : int | |
| 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.   |  | | --- | | **InvalidHours Class** | | − value : int | | + InvalidHours(h : int) :  + getValue() : int | |
| 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.   |  | | --- | | **InvalidEnglish Class** | | − value : int | | + InvalidEnglish(l : int) :  + getValue() : int | |
| 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.   |  | | --- | | **InvalidHealth Class** | | − value : int | | + InvalidHealth(h : int) :  + getValue() : int | |
| getFlightHours | Public | The member function returns the value of the **flightHours** member variable. |
| getHoursInCommand | Public | The member function returns the value of the **hoursInCommand** member variable. |
| getEnglishLevel | Public | The member function returns the value of the **englishLevel** member variable. |
| getHealthStatus | Public | The member function returns the value of the **healthStatus** member variable. |
| 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, hours in command, English 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, hours in command, English proficiency level, and health status. |

## The PilotCertificate class:

The PilotCertificate class holds official information about the pilot’s license including the license type 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 exceptions, which is InvalidType, when the license type is invalid. 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 member variables and member functions 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 first removes spaces and capitalizes the argument, then it checks if the argument is a valid type of license. If the argument is invalid, then the function throws an InvalidType object with invalid data as an exception. Otherwise, the function stores the argument in **licenseType**.   |  | | --- | | **InvalidType Class** | | − type : string | | + InvalidType(t : string) :  + getType() : string | |
| 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. |
| getExpiryDate | Public | The member function returns the expiration date of the pilot’s license by returning a Date object. |
| operator << | Public | The overloaded operator prints details of the pilot’s certificate out. |
| operator >> | Public | The overloaded operator gets input of a PilotCertificate object including license type and expiry date. |

## 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 exceptions, when the input data is invalid.

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 level, 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 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.   |  | | --- | | **InvalidName Class** | | − value : string | | + InvalidName(n : string) :  + getValue() : string | |
| 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. It returns a PilotCompetence object. |
| getPilotCertificate | Public | The member function returns the certificate of the pilot. It returns a PilotCertificate object. |
| operator << | Public | The overloaded operator prints details of the pilot’s information out. |
| operator >> | Public | The overloaded operator gets input of a Pilot object including name, competence, and certificate. |

## 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  + setDate(date : string) : void  + isLeapYear() : bool  + operator > (date : Date) : bool  + operator >> (strm : istream &, obj : Date) : istream &  + operator << (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 assign 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 | |
| setDate | Public | The member function accepts a string of a date in the format MM/DD/YYYY or MM-DD-YYYY, parses the string into month, day, year, and calls mutator functions to set the month, day, and year. |
| 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. |

## 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 hold the 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 of the **title** member variable. |
| getInspectionResult | Public | The member function returns the value in the **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 aspect has different criteria to inspect. |
| ~InspectionResult | Public | The default virtual destructor will ensure that the destructors of the derived classes are called when using polymorphism. |

## 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 are 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 are met. 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 **setInspectionResult** 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. |
| setLicenseExpiryResult | 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 the **flightHoursResult** member variable. |
| getHoursInCommandResult | Public | The member function returns the value in the **hoursInCommand** member variable. |
| getEnglishLevelResult | Public | The member function returns the value in the **englishLevelResult** member variable. |
| getHealthStatusResult | Public | The member function returns the value in the **healthStatusResult** member variable. |
| getLicenseTypeResult | Public | The member function returns the value in the **licenseTypeResult** member variable. |
| getLicenseExpiryResult | Public | The member function returns the value in the **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 are called when using polymorphism. |

## 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. |

# Input validation:

## Input validation for the Date class

We assume that the user enters the data in the format MM DD YYYY or M D YYYY.

The value of month, day, and 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 throws an exception. If the value of the day is invalid, then the function throws an exception. If the value of the year is invalid, then the function throws an exception.

* + - 1. **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.

* + - 1. **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.

* + - 1. **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.

# B. Plane-related classes and documentation:

# C. Weather-related classes and documentation:

Vu Tien Tue – 1694603

**1. Weather standards that we support**

As we said before, our program mainly support flights of Vietname Airlines which take off from Vietnam. Therefore, I design weather inspection simulation based on Vietnamese weather standards from official sources that I found out. In practice, the required weather qualities which allow a flight to depart can vary on different altittude, but I synthesize a general standard to deploy our Program

|  |  |
| --- | --- |
| **Vietnameese Weather Standard** | |
| Minimum forward Visibility | 250m |
| Minimum horizontal Visibility | 6000m |
| Maximum Crosswind | 5 knots |
| Maximum Tailwind | 10 knots |
| Maximum Temperature | 50 Celcius degree |
| Minimum Temperature: | -47 Celcius degree |
| Maximum distance to thunderstorm | 5 nautical miles |

**2. Weather Standard classes contain weather standard**

**2.1 Class WeatherStandard**

I firstly design a blueprint, which perform as an international weather standards pattern, for general weather standard cases. Due to the difference in climate between countries, We can base on this pattern to build weather standards for each countries if we want to develop a program that support internaltional departure. The UML for this class is below:

|  |
| --- |
| **WeatherStandard Class** |
| + visibility : float  + horizontalVisibility : float  + crosswind : float  + tailwind : float  + temperatureUpperBound : float  + temperatureLowerBound : float    + thunderstorm : float    + freezing : bool |
| WeatherStandard(): Default constructor  WeatherStandard(float visibility, float horizontalVisibility, float crosswind,float tailwind, float temperatureUpperBound, float temperatureLowerBound, float thunderstorm): parameterized constructor      // /Getters and setters      float getTailwind() const;      float getVisibility() const;      float getHorizontalVisibility() const;      float getCrosswind() const;      float getTemperatureUpperBound() const;      float getTemperatureLowerBound() const;      float getThunderstorm() const;      bool getFreezing() const;      void setTailwind(float tailwind);      void setVisibility(float visibility);      void setHorizontalVisibility(float horizontalVisibility);      void setCrosswind(float crosswind);      void setTemperatureUpperBound(float temperatureUpperBound);      void setTemperatureLowerBound(float temperatureLowerBound);      void setThunderstorm(float thunderstorm);      void setFreezing(bool Freezing);      //Check whether the weather is qualified      virtual bool isWeatherAcceptable(const Weather& weather) const ; |

|  |  |  |
| --- | --- | --- |
| **Members** | **Access mode** | **Description** |
| visibility | Private | Minimum forward Visibility |
| horizontalVisibility | Private | Minimum horizontal Visibility |
| crosswind | Private | Maximum crosswind |
| tailwind | Private | Maximum tailwind |
| temperatureUpperBound | Private | Maximum temperature |
| temperatureUpperBound | Private | Maximum temperature |
| thunderstorm | Private | Maximum distance to thunderstorm |
| freezing | Private | True if rưnay is freeze that can make taking off dangerous, and false if vice versa |
| WeatherStandard() | Default Constructor | Initialize above attributes as 0 |
| WeatherStandard(float visibility, float horizontalVisibility, float crosswind,float tailwind, float temperatureUpperBound, float temperatureLowerBound, float thunderstorm): | Parameterized  Constructor | Initialize above attributes as parameters, except for freezing |
| getTailwind  getVisibility  getHorizontalVisibility()  getCrosswind  getTemperatureUpperBound getTemperatureLowerBound  getThunderstorm  bool getFreezing   setTailwind   setVisibility   setHorizontalVisibility  setCrosswind  setTemperatureUpperBound  setTemperatureLowerBound  setThunderstorm  setFreezing | Public | Setters which assign the value, and getters that get the value of above attributes: tailwind, visibility, horizontalvisibility,crosswind, temperatureUpperBound, temperatureLowerBound, thunderstorm, freezing |
| isWeatherAcceptable | Public | Check if current weather is acceptable to take off |

**2.2 Class WeatherStandardVN**

* This class inherits WeatherStandard class; howerver due to Vietnamese climate condition, I eliminate freezing condition( remove the attribute: bool freezing)
* Default constructor: WeatherStandardVN() initializes Vietnamese standards that I give you in **1. Weather standards that we support** as follows:

+) visibility = 250.0

+) horizontalVisibility = 5000.0

+) crosswind = 15.0

+) tailwind = 10.0

+)  temperatureUpperBound = 50.0

+) temperatureLowerBound = 47.0

+) thunderstorm = 5.0

* Parameterized Constructor: allowing user to initialize custom qualities

**3 Class Weather**

Weather class is designed for objects that contains current weather information

|  |
| --- |
| **Weather Class** |
| + float visibility : the current forward visibility  + float horizontalVisibility : the current horizontal visibility  + float crosswind : the current crosswind consituon  + float tailwind : the current tailwind consituon  + float temperature : the current temperature condition    + float thunderstorm : current distance to thunderstorm    + bool freezing: the current freezed condition of runway |
| Weather(): Default constructor which initialize every attributes equals to 0  WeatherStandard(float visibility, float horizontalVisibility, float crosswind,float tailwind, float temperatureUpperBound, float temperatureLowerBound, float thunderstorm): parameterized constructor that initializes attributes following parameters   * Getters that get value, and setters that assign above attributes: visibility, horizontalVisibility, crosswind, tailwind, temperature, thunderstorm, and freezing       float getVisibility() const;      float getCrosswind() const;      float getTemperature() const;      float getThunderstorm() const;      float getTailwind() const;      float getHorizontalVisibility() const;      bool getFreezing() const;      void setVisibility(float visibility);      void setCrosswind(float crosswind);      void setTemperature(float temperature);      void setThunderstorm(float thunderstorm);      void setTailwind(float tailwind);      void setHorizontalVisibility(float horizontalVisibility);      void setFreezing(bool Freezing);  ostream& operator<<(ostream& os, const Weather& weather):   * Operator overloading << : printing out the current Vietnamese weather information of Weather object( do not display the freezing condition) * Operator overloading >> : allowing user to input current Vienamese weather( not allowing to input freezing condition) |

**4 class WeatherInspectionResult**

This class contains result of inspection. In specific, Each attributes represents the result of checking each standards, and there is a attribute which represents the overall inspection result

|  |
| --- |
| **WeatherInspectionResult Class** |
| + bool isVisibility : boolean value represents whether the visibility is eligible  + bool horizontalVisibility : boolean value represents whether the horizontalVisibility is eligible  + bool crosswind : boolean value represents whether the crosswind is eligible  + bool tailwind : boolean value represents whether the crosswind is eligible  + bool temperature : boolean value represents whether the crosswind is eligible  + float thunderstorm : current distance to thunderstorm  + bool inspectionResult : overall result of inspection |
| WeatherInspectionResult();      WeatherInspectionResult(bool isVisibility,bool isHorizontalVisibility, bool isCrosswind, bool isTailw, bool isTemperature, bool isThunderstorm);      //Getters      bool getIsVisibility() const;      bool getIsCrosswind() const;      bool getIsTemperature() const;      bool getIsThunderstorm() const;      bool getIsTailwind() const;      bool getIsHorizontalVisibility() const;      bool getInspectionResult() const;      //Setters      void setIsVisibility(bool isVisibility);      void setIsCrosswind(bool isCrosswind);      void setIsTemperature(bool isTemperature);      void setIsThunderstorm(bool isThunderstorm);      void setIsTailwind(bool isTailwind);      void setIsHorizontalVisibility(bool isHorizontalVisibility);      void setInspectionResult(bool inspectionResult); |

|  |  |  |
| --- | --- | --- |
| **Members** | **Access mode** | **Description** |
| WeatherInspectionResult(); | Default constructor | Initializing every stardanrds to zero |
| WeatherInspectionResult(bool isVisibility,bool isHorizontalVisibility, bool isCrosswind, bool isTailw, bool isTemperature, bool isThunderstorm); | Parameterized constructor | Initializing object of WeatherInspectionResult, allowing you to adjust the status of each quality |
| //Getters  getIsVisibility()  getIsCrosswind()  getIsTemperature()  getIsThunderstorm()  getIsTailwind()  getIsHorizontalVisibility()  getInspectionResult() const;  //Setters  setIsVisibility()  setIsCrosswind()  setIsTemperature()  setIsThunderstorm()  setIsTailwind()      void setIsHorizontalVisibility(bool isHorizontalVisibility);      void setInspectionResult(bool inspectionResult); | Public | Setters which assign the bool value, and getters that get the value of above attributes |

# D. Universal classes:

# Flight class

The class holds data about a flight. Each member in the group will design the functions corresponding to their roles.

* Ly Ba Hoang (1695172): is responsible for member variables **pilot**, **pilotInspectionResult**, member functions **setPilot**, **setPilotInspectionResult**, **getPilot**, and **getPilotInspectionResult**.
* Vu Tien Tue (16946043) : is in charge of attributes **weather, weatherInspectionResult**, methods **getWeather, getWeatherInspectionResult**

|  |
| --- |
| **Flight Class** |
| # pilot : Pilot  # pilotInspectionResult : PilotInspectionResult  # weather: Weather  # weatherInspectionResult: WeatherInspectionResult |
| + setPilot(pilotInfor : Pilot) : void  + setPilotInspectionResult(result : PilotInspectionResult) : void  + getPilot() : Pilot  + getPilotInspectionResult() : PilotInspectionResult |

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

# FlightInspection class

The class performs inspection. It performs inspection on pilot, plane, and weather. Each member in the group will design the inspection function corresponding to their roles.

* Ly Ba Hoang (1695172): is responsible for defining functions that perform inspection on the pilot including **inspectPilot** function.

Here is the UML of the class:

|  |
| --- |
| **FlightInspectionDepartment Class** |
|  |
| {static} + inspectPilot(pilotInfo : Pilot, standard : PilotStandard) : PilotInspectionResult  {static} + inspectWeather( flight : Flight, weatherStandard : WeatherStandardVN |

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, the functions check each criteria of a pilot including flight hours, hours in command, license type, English level, health status, license expiration, and returns a PilotInsepctionResult object, which contains all the details of the inspection result. |
| InspectWeather | Public | This method receive 2 parameters: a Flight object and WeatherStandardVN object. This method get information of forecast weather of this flight through flight.getWeather. Then WeatherStandardVN object will check the forecast weather of the flight   * If true, using setInspectionResult to update inspectionResult as true, the same for detail standards that they will also be updated as true * If false, set inspectionResult as false, and check each detail standards and update status of them |

## FlightManagement class

The class is responsible for holding pilot standards, loading pilot standards from the file, searching for the pilot standards based on the model, and writing the vector of flights to files. Each member in the group will design the functions corresponding to their roles.

* Ly Ba Hoang (1695172): is responsible for the member variable **pilotStandardArray**, the member functions **loadPilotStandard**, **findPilotStandard**, and **displayPilotStandards**.

|  |
| --- |
| **FlightManagement 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 objects. |
| loadPilotStandard | Public | The member function accepts the name of a file 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 returns the corresponding PilotStandard object. |
| displayPilotStandards | Public | The member variable displays all the PilotStandard objects in the **pilotStandardArray** vector. |

# 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>