COMP2404B (Fall 21) -"Introduction to Software Engineering"

Assignment 4 of 4 - Due Sunday March 21th, 11:59 pm

- Your assignment should be submitted online on cuLearn as a single .zip file named "a1.zip". Select all your files, right-click → "compress" and select ".zip" Files in other formats will be rejected and will receive a mark of zero.
- No late tutorials will be accepted.
- Pay close attention to the public interface. That means that
  - function names
  - function return types
  - parameter types and order
  - class definitions

  should all be **EXACTLY** as described. If the script can't read it, you get 0 for that part.
- Your code should be neat, readable, and documented. In the event that myself or a TA must mark this manually, you WILL be evaluated on how easy the code is to understand.
- In addition to the items covered by the marking script, additional items may be marked. This also includes non-functional requirements (for example, we can check using a script whether you have forward referenced a function).

---

1 Application Description

In this assignment you will be writing a parts control system for an airline using C++.

An `Airline` runs a number of `Aircraft`. Spare `Part`s for the aircraft are stored in case a part must be changed. The rules and regulations of the airline industry ensure that parts are inspected and changed at regular intervals. As such, each `Part` is carefully tracked. Some parts must be inspected after a certain number of flight hours. Some parts must be inspected after a certain amount of time being installed in an aircraft. And some parts require both flight hours and time installed to be tracked. This behaviour is the perfect opportunity to use multiple inheritance!

`Part`s can be installed on an `Aircraft` on a given `Date`. Once installed, every time the aircraft flies the number of flight hours are recorded on the `Part`. The airline may choose to do an `inspectionReport` on a particular aircraft on a given `Date`, at which time all the parts that are due for inspection are reported.

2 Learning Outcomes

In this assignment you will learn

1. Multiple inheritance and the diamond hierarchy.
2. Polymorphism
3. Operator overloading (use `ostream <<` instead of `print()` when instructed).
4. Templates (in the form of an `Array` class for storing different data types).

3 Classes Overview

This application will consist of 8 classes. In addition to the classes shown in the diagram above, there is a `PodArray` class. The classes are listed below along with their respective categories.

1. `Date` (Entity object):
   (a) Stores information of a date.
2. `Part` (Entity object):
   (a) Virtual base class of all `Part`s.
3. `FH_Part` (Entity object):
   (a) A part that must be inspected after a certain number of flight hours.
4. `IT_Part` (Entity object):
   (a) A part that must be inspected after a certain number of days of being installed on an aircraft.
5. `FHIT_Part` (Entity object):
   (a) A part that must be inspected after a certain number of days of being installed on an aircraft and after a certain number of flight hours.
6. `Array` (Container object):
   (a) A simple (templated) data structure.
7. `Aircraft` (Entity / Container object):
   (a) Aircraft data as well as a container for the installed `Part`s.
8. `Airline` (Control object):
   (a) Tracks parts, aircraft, installations, flights, etc.
9. `Control` (Control object):
   (a) Provides a test framework for the `Airline`.
10. `View` (View object):
   (a) Collects user input and provides system output.

4 Instructions

All member variables should be given the appropriate access modifier (hint: it isn't `public`). All member functions are `public` unless otherwise noted. Some return values are not explicitly given. You should use your best judgment (they will often be `void`, but not always). ALL CLASSES MUST HAVE A PRINT FUNCTION OR OVERLOAD THE << OPERATOR (AS INSTRUCTED). This should display the metadata of the class using appropriate formatting.

4.1 The Date Class

This class is given to you. Take note of the fact that it overloads the << operator rather than having a `print` function. Also take note of the `toDays` function. This converts the `Date` into the number of days since January 1st, 1901. You can use this function to determine the number of days between two `Date`s.

4.2 The Part Class

Implement the `Part` base class. **Note**: You may, in the interests of keeping the number of files to a manageable level, combine the `Part`, `FH_Part`, `IT_Part`, and `FHIT_Part` headers into one file "Part.h", and all of the source files into one file "Part.cc".

1. Member variables:
   (a) `string name`: The name and unique identifier of the `Part`.
   (b) `Date installationDate`: The date this part was installed on an aircraft.
   (c) `int flighthours`: The number of hours this part has flown installed on an aircraft.
2. Make a getter for the part name.
3. Make a one argument constructor that takes a string and uses it to initialize the name member variable.
4. Make an appropriate destructor if one is needed.
5. Member functions.
   (a) `void addFlightHours(int)`. Add the given number to the flight hours member variable.
   (b) `void install(Date&)`. Update the `installationDate` member variable with the given date.
6. Instead of a print function, overload the << operator for `ostream`s.
7. Pure virtual member functions.
   (a) Make a function `inspection`. It should take one `Date` argument which is the date of the inspection. This function should return true if the part is due for inspection, and false otherwise. The subclasses will implement the logic to determine if the part requires inspection.

4.3 The FH_Part Class

Should inherit from the `Part` class. The `FH` stands for "flight hours".

1. Member variables:
   (a) `int fh_inspect` - the number of flight hours until inspection is required.
2. Make a constructor that takes a name and a number and initializes name and `int fh_inspect`.
3. Make a destructor if needed.
4. Inherited virtual member functions.
   (a) `virtual bool inspection`: return true if the flightHours is greater than or equal to the `fh_inspect` variable and false otherwise.

4.4 The IT_Part Class

Should inherit from the `Part` class. The `IT` stands for "installed time".

1. Member variables:
    (a) `int it_inspect` - the number of days installed ("install time") until inspection is required.
2. Make a constructor that takes a name and a number and initializes name and `int it_inspect`.
3. Make a destructor if needed.
4. Inherited virtual member functions.
    (a) `virtual bool inspection`: return true if the number of days between the installation date and the inspection date is greater than or equal to the `it_inspect` variable and false otherwise.

4.5 The FHIT_Part Class

Should inherit from the `FH_Part` and `IT_Part` classes.

1. Make a constructor that takes a name and two integers and initializes name, `int fh_inspect` and `int it_inspect`.
2. Make a destructor if needed.
3. Inherited virtual member functions.
    (a) `virtual bool inspection`: return true if either of `FH_Part::inspection` and `IT_Part::inspection` return true or false otherwise.

4.6 The Array Class

The array class is provided for you, but it currently only works for `int`s. Make the following modifications. Also note that both the class definition and implementation are in the same .h file.

1. Convert the `Array` class to a Template class so that it may store any type.
2. Change the `get` method into a method that overloads the `[]` operator.
3. Overload the `ostream <<` operator so that the `Array` writes to `ostream` all of the contained elements. Note that if you choose to store pointers in the array and use the `ostream <<` operator it should print out a series of memory locations, which looks terrible, but is the behaviour that we want.

4.7 The Aircraft Class

1. Member variables:
    (a) `string type`: The type of aircraft.
    (b) `string registration`: This is the unique identifier of the aircraft (it is the string of letters and numbers generally seen close to the tail of the aircraft).
    (c) `int flighthours`: The number of hours this aircraft has flown.
    (d) An `Array` of `Part` pointers.
2. Make a getter for the registration.

3. Make a two argument constructor that takes two strings for the type and registration member variables.
4. Make an appropriate destructor if one is needed.
5. Member functions.
   (a) `void install(Part*, Date&)`. Add the given `Part*` to the `Array`. Call the appropriate member function on the `Part` object to install the part.
   (b) `void takeFlight(int hours)`. Update the `flightHours` member variable by adding the supplied number of `hours`. This should also update the `flighthours` of all `Parts` installed in the aircraft.
   (c) `inspectionReport`. This should have a `Date` input parameter and an `Array<Part*>` output parameter. Iterate over all `Parts` installed on the plane. Use the `inspection` function to determine if an inspection is required. If an inspection is required on a `Part`, add that `Part` to the supplied output `Array`.
6. Instead of a print function, overload the << operator for `ostreams`.

4.8 The Airline Class

1. Member variables:
   (a) `string name`: The name of the airline.
   (b) An `Array` of `Part` pointers.
   (c) An `Array` of `Aircraft` pointers.
2. Make a constructor that takes a `string& name` as an argument. Initialize all member variables.
3. Make a destructor. The `Airline` is responsible for deleting all `Aircraft` and `Parts` in its `Array`.
4. Member functions:
   (a) Make getters for `Aircraft` and `Parts`. These are helper functions and thus should be `private`. Both of the getters should accept a `string` as an input variable. The `Aircraft` getter should return the `Aircraft` with the matching `registration`. The `Part` getter should return the `Part` with the matching `name`.
   (b) Make an `addAircraft` function. This function should take a `string` type and a `string` registration as arguments. It should use these arguments to make a new `Aircraft` object and add it to the appropriate `Array`.
   (c) Make an `addPart(const string& part, int fh_inspect, int it_inspect)` function. This function should use the arguments to determine the appropriate `Part` to make. If `fh_inspect` equals 0, the make an `IT_Part`. If `it_inspect` equals 0, the make an `FH_Part`. If neither is 0, make an `FHIT_Part`. Add this part to the appropriate `Array`.
   (d) Make a `takeFlight` function that accepts a `string reg` and an `int hours` as arguments. It should retrieve the `Aircraft` with the given registration (if it exists) and have it `takeFlight` for the given number of hours.
   (e) Make a `printAircraft` and `printParts` method. These should print all the `Aircraft` and all the `Parts` in the `Airline` respectively.
   (f) Make an `inspectionReport` function. This should take as arguments a `string reg` and a `Date&`. Find the `Aircraft` with the given registration and print out (to `cout`) the `Aircraft` registration and all the parts on that `Aircraft` that require inspection.
   (g) Make an `bool install` function that takes two `strings` and a `Date&` as arguments. The first string is an `Aircraft registration` and the second string is a `Part name`. If there is a an `Aircraft` with the given `registration` and a `Part` with the given `name` in the `Airline`, install that `Part` into the `Aircraft` and return `true`. You do not have to check if it is already installed on an `Aircraft` - we will assume the necessary error checking for that is done elsewhere. If the `Aircraft` or the `Part` does not exist in the `Airline`, return `false`.

4.9 The Control Class

This class is provided for you. The `launch` function instantiates and displays a `View` object to gather user input. Based on the user input, it calls the necessary `Airline` functions.

4.10 The View Class

This class is provided for you. You may make minor changes if you wish, or add options, but please leave the current options 1-4 as is.

4.11 The `main` Function

Provided for you. It instantiates and `launch`es a `Control` object.

5 Constraints

Your program must comply with all of the rules of correct software engineering that we have learned during the lectures, including but not restricted to:

1. The code must compile and execute in the default course VM provided. It must NOT require any additional libraries, packages, or software besides what is available in the standard VM.
2. Your program must not use any classes, containers, or algorithms from the standard template library (STL) *unless expressly permitted.*
3. Your program must be written in Object-Oriented C++. To wit:
   (a) Do not use any global functions or variables other than `main`.
   (b) Do not use `struct`s, use classes.
   (c) Do not pass *objects* by value. Pass by reference or by pointer.
   (d) Except for simple getters or error signalling, data must be returned from functions using output parameters.
   (e) Reuse existing functions wherever possible. If you have large sections of duplicate code, consider consolidating it.
   (f) Basic error checking must be performed.
   (g) All dynamically allocated memory must be deallocated. Every time you use the `new` keyword to allocate memory, you should know exactly when and where this memory gets `delete`d. Use `valgrind`.
4. All classes should be reasonably documented (remember the best documentation is expressive variable and function names, and clear purposes for each class).

6 Submission

7 Grading

7.1 Marking Components

1. 15 marks: Correct implementation of `Part`.
2. 30 marks: Correct implementation of `FH_Part`, `IT_Part`, and `FHIT_Part` classes.

3. 10 marks: Correct implementation of `Array` class.
4. 10 marks: Correct implementation of `Aircraft` class.
5. 20 marks: Correct implementation of `Airline` class.

Total: 85 marks.

7.2 Execution and Testing Requirements

1. All marking components must be called and execute successfully to earn marks.
2. All data handled must be printed to the screen to earn marks (make sure `print` prints useful information, such as the object member variables, where appropriate).

7.3 Deductions

7.3.1 Packaging errors:

1. 10 marks: Missing Makefile
2. 5 marks: Missing README
3. up to 10 marks: Failure to separate code into header and source files.
4. up tp 10 marks: Readability - bad style, missing documentation.

7.3.2 Major design and programming errors:

1. 50%: marking component that uses global variables or `struct`s.
2. 50%: marking component that consistently fails to use correct design principles.
3. 50%: marking component that uses prohibited library classes or functions.
4. up to 10 marks: memory leaks reported by `valgrind`.

7.3.3 Execution errors:

1. 100% of any marking component that cannot be tested because it doesn't compile or execute in the course VM, or the feature is not used in the code, or data cannot be printed to the screen. In short: your program must convince, without modifcation, myself or the TA that it works and works properly. TAs are not required to debug or fix non-working code.