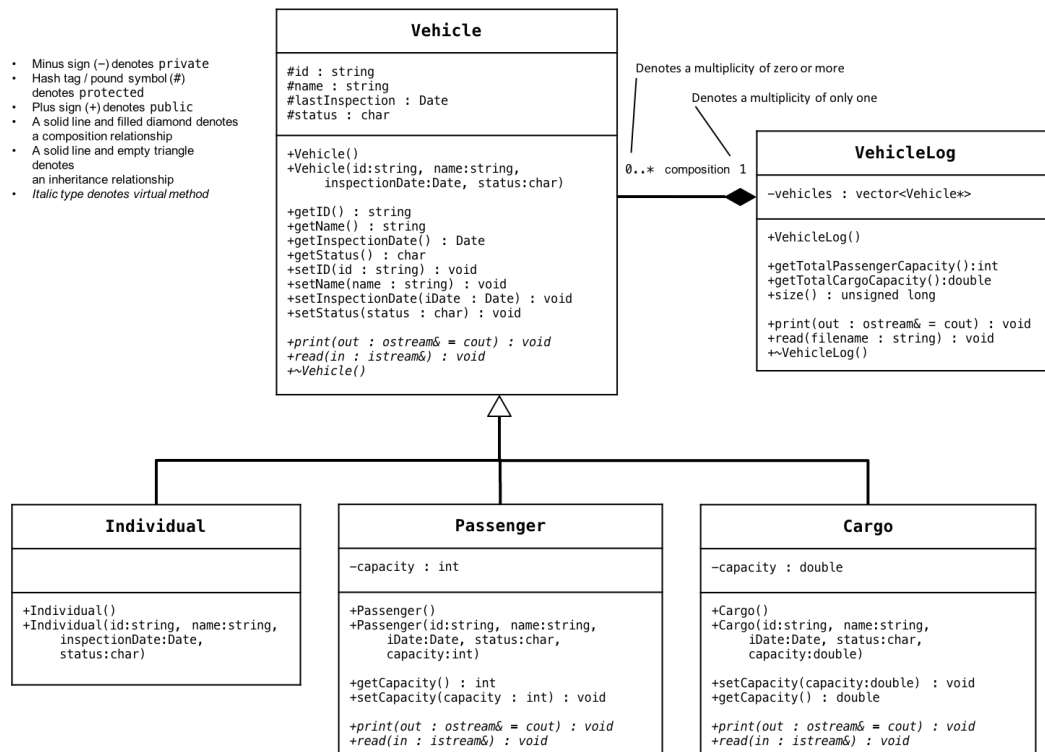


Background

You are a member of a software development team that has been hired to develop an application to support a vehicle rental company. Bubba Hotep, regional manager for the Northeastern United States, has historically kept all vehicle records in a paper log book (most of which he could also recall from memory). Unfortunately, Bubba's log book was totally destroyed by flooding during Hurricane Ida. To ensure that this does not happen in the future, and to reduce dependence upon Bubba's freakish memory, the company wants all records digitized and automated. In this project, you will use classes, inheritance, and polymorphism to implement software that stores information about different types of rental vehicles. Vehicles are one of three types: **Individual**, **Passenger**, or **Cargo**. The Unified Modeling Language (UML) diagram below is provided for reference in completing the project.



Computer scientists and software engineers commonly use UML to design software and to communicate those designs. We will discuss the diagram in class, but its interpretation should be relatively straightforward. A rectangle represents a class and is divided into three sections for the class' name, its data members (or attributes or properties), and its methods (or operators). Characters precede attributes and methods. The (+) sign denotes public, the hyphen (-) denotes private, and the pound sign (#) denotes protected. Many additional details and validation rules are also available at:

<http://people.cs.georgetown.edu/~addison/projects/fall2021/p2docs/index.html>

As the diagram indicates, you should implement the **Vehicle** class and use inheritance to derive three subclasses: **Individual**, **Passenger**, and **Cargo**. All vehicles have an `id`, a `name`, an `inspectionDate`, and a `status`. These are protected in the base class. **Passenger** vehicles have a `capacity` that is the maximum number of passengers. **Cargo** vehicles have a `capacity` that is the maximum payload in tons. Each class should have virtual methods that read and print the vehicle's information from and to a stream. Naturally,

the setter methods should perform validation checks and throw an exception for values that are outside of limits or otherwise not valid.

We shall also implement a `VehicleLog` class that stores pointers to `Vehicle` objects in a C++ vector. As the composition link in the diagram indicates, there is a one-to-many relationship between the `VehicleLog` class and the `Vehicle` class. Implement methods to read `Vehicle` objects from a file, print the `Vehicle` objects stored in the vector (Note that elements stored in the vector are actually pointers to `Vehicle` objects), compute the total passenger carrying capacity of all of the `Passenger` vehicles, and compute the total cargo hauling capacity of the `Cargo` vehicles.

The input data file format consists of a `Vehicle` object's `id` (a string without spaces), `name` (vehicle make and model, a string with spaces), `inspectionDate` (a `Date` object). `Passenger` vehicle records list a `capacity`, an integer, that is the maximum number of passengers the vehicle can carry. `Cargo` vehicle records also list a `capacity`, a floating-point number, that is the maximum payload hauling capacity in tons. `Individual` vehicle records do not have a `capacity`, so there is one less data element to read from those rows of the file. The `id` is a string of exactly 17 characters. The first character is a letter indicating the type of the vehicle. This character is 'i' for `Individual` vehicles, it is 'p' for `Passenger` vehicles, and it is 'c' for `Cargo` vehicles. The remaining 16 characters are an alpha-numeric vehicle identification number (VIN). The `name` is a string containing spaces, so it is enclosed within double quotes. The `inspectionDate` is a string with no spaces in our standard date format (yyyy/mm/dd). The `status` is a single character. This character is 'm' indicating the vehicle is in maintenance; 's' indicating the vehicle is in service, or in use; 'a' indicating the vehicle is available to rent. The `capacity` is either an integer or a floating-point number depending upon the vehicle type (remember, that there is no `capacity` for `Individual` vehicles). Below is a small sample of input data.

p1GCGG25C39110652	"Chevrolet Express Passenger"	2013/07/18	s	15
c1GCGG25C48112351	"Ford F 250"	2007/02/01	s	1.9
p1GCGG25C59114279	"Ford XL Passenger"	2012/02/10	s	15
c1GCGG25C68118226	"Chevrolet Silverado 2500"	2016/10/15	s	1.62
p1GCGG25C68120131	"Ford XL Passenger"	2016/01/06	s	15
c1GCGG25C69110536	"Ford Super Duty 450 XL"	2008/02/23	s	3.82
i1GCGG25CX8119027	"Dodge Demon"	2015/09/24	s	
c1GCGG25R5W102803	"Dodge Ram 2500"	2010/06/05	s	1.53
i1GCGG25V07122904	"Dodge Demon"	2016/08/12	s	
i1GCGG25V07124420	"Ford Explorer"	2011/10/12	s	
i1GCGG25V14110105	"Ford EcoSport"	2017/01/22	s	

As the method `VehicleLog::read` reads the input file, it must decide to instantiate either an `Individual`, `Passenger`, or `Cargo` vehicle object based upon the contents of the input stream. Once the method `VehicleLog::read` determines whether the information in the stream corresponds to an `Individual`, `Passenger`, or `Cargo` vehicle it shall dynamically allocate memory for, and instantiate an object of the appropriate derived class. After that, `VehicleLog::read` shall invoke the overwritten `read` method of, the derived object to complete initialization of its data members from the stream contents. The member function `VehicleLog::read` must determine which derived class to instantiate based upon the value of the first character of the `id`. You will need a strategy to determine the value of the first character of the next row of data without modifying the contents of the stream.

Your driver program shall accept the full path and name of the input data file as a command line argument. In function `main`, write code that instantiates a `VehicleLog` object, reads vehicles from the input data file `vehicleList01.txt`, prints all valid objects (those appended to the vector) to the terminal window in the *external format* (see below), prints the total passenger carrying capacity of **all Passenger** vehicles, and prints the total payload hauling capacity of **all Cargo** vehicles. Function `main` shall also catch any exceptions that could be thrown during the program execution and display an appropriate message for those exceptions.

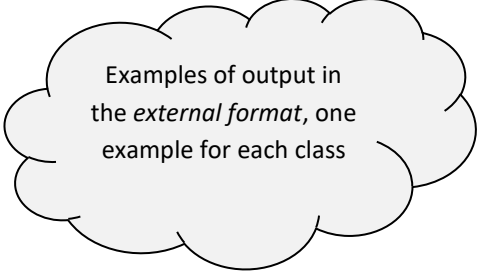
```

Vehicle ID:   i19XFB2F58CE07064
Make/Model:  Dodge Challenger
Last Inspection: 2014/11/11
Current Status: In Maintenance

Vehicle ID:   p1G1AL52F95752778
Make/Model:  Chevrolet Express Passenger
Last Inspection: 2016/03/16
Current Status: In Service
Max. Passengers: 15

Vehicle ID:   c5TDZARFH8ES00485
Make/Model:  Ford F 250
Last Inspection: 2015/04/26
Current Status: Available
Max. Payload: 1.90 tons

```



Examples of output in the *external format*, one example for each class

Getting Started

Several files are attached to the Project 2 Assignment on Canvas to help get you started. You may also copy all `.h` files from the on-line documentation. You should create a directory on your server account named `p2` and copy all of these files to that directory. The `p2` directory will then contain a `Makefile` and `.h` files for the project. If you are developing via an IDE, do the same thing on your local computer and then add the `.h` files to your IDE project. I recommend that you immediately write function stubs for **all** methods of **all** classes. Thereafter, use stepwise refinement and incremental development. For example, write implementation code for `Vehicle` class methods and test those methods thoroughly before implementing derived class member functions.

Submission Details

What to submit: One compressed file containing all source code and any other files associated with this project. The file name should be `submit.zip`. You must separate your class specification details from your class implementation details. Therefore, you must prepare header files (`<filename>.h`) and associated implementation files (`<filename>.cpp`). Ensure that your `.h` files contain sufficient comments for each data member and class method. Additionally, you must provide another `.cpp` file that contains function `main()`, along with its associated `.h` file. This "driver" program is where class objects are instantiated and functionality of the software is demonstrated. Use these names with **spelling and capitalization exactly as shown**:

```

Resources.h, Resources.cpp (reuse from P1)
Exceptions.h, Exceptions.cpp
Vehicle.h, Vehicle.cpp
VehicleLog.h, VehicleLog.cpp

```

```

main.h
main.cpp
Makefile

```

Due date/time: 7 October 2021, no later than end-of-day (11:59pm). Late submissions will be penalized 2.5 points for each 15 minutes late. If you are over 10 hours late you may turn in the project to receive feedback but the grade will be zero. In general requests for extensions will not be considered.

Creating submit.zip: **Please, PLEASE, PLEASE** use the provided Makefile and create your submit.zip file on the class server. If you create the compressed file on your laptop it is highly likely something will go wrong even though it looks fine. It is easy to compress links to files, instead of actual files. It is easy to have the folder containing the project files included in the compressed file. If this, or any other problems happen; your program will not compile, automated grading programs will fail, **and you will get a zero**. Assuming all of your files are in the same folder on the server, the process to create the submit.zip file is shown below.

```
[waw23@cs-class-1 P2]$ make clean
rm -f *.o core a.out
[waw23@cs-class-1 P2]$ make submit
rm -f submit.zip
zip submit.zip main.cpp main.h VehicleLog.cpp VehicleLog.h Vehicle.cpp Vehicle.h
Exceptions.cpp Exceptions.h Resources.cpp Resources.h Makefile
  adding: main.cpp (deflated 68%)
  adding: main.h (deflated 45%)
  adding: VehicleLog.cpp (deflated 73%)
  adding: VehicleLog.h (deflated 64%)
  adding: Vehicle.cpp (deflated 81%)
  adding: Vehicle.h (deflated 79%)
  adding: Exceptions.cpp (deflated 72%)
  adding: Exceptions.h (deflated 84%)
  adding: Resources.cpp (deflated 84%)
  adding: Resources.h (deflated 73%)
  adding: Makefile (deflated 67%)
[waw23@cs-class-1 P2]$ unzip -l submit.zip
Archive:  submit.zip
  Length      Date    Time    Name
-----
    5774  09-19-2021  21:24   main.cpp
     776  09-19-2021  21:24   main.h
   12178  09-19-2021  21:24  VehicleLog.cpp
   3141   09-19-2021  21:24  VehicleLog.h
   27930  09-19-2021  21:24  Vehicle.cpp
   10520  09-19-2021  21:24  Vehicle.h
    2488  09-19-2021  21:24  Exceptions.cpp
    7649  09-19-2021  21:24  Exceptions.h
   43823  09-19-2021  21:24  Resources.cpp
    9108  09-19-2021  21:24  Resources.h
    1140  09-19-2021  21:29  Makefile
-----
   124527                     11 files
[waw23@cs-class-1 P2]$
```

Remove files from last compile

Create the zip file to submit

Verify the zip file contents, make sure the date and time are correct and these are the files you want to submit, you may make unlimited submissions prior to the due date, the last submission will be graded

Academic Integrity

This is an individual project and all work must be your own. Refer to the guidelines specified in the *Academic Honesty* section of this course syllabus or contact me if you have any questions.

Include the following comments (with appropriate substitutions) at the start of each file in your project:

```
/*
 * <FileName>.<file extension>
 *
 * COSC 052 Fall 2021
 * Project #2
 *
 * Due on: OCT 7, 2021
 * Author: <your name> <your netID>
 *
 *
 * In accordance with the class policies and Georgetown's
 * Honor Code, I certify that, with the exception of the
 * class resources and those items noted below, I have neither
 * given nor received any assistance on this project.
 *
 * References not otherwise commented within the program source code.
 * Note that you should not mention any help from the TAs, the professor,
 * or any code taken from the class textbooks.
 */
```

Grading

This graded assignment is worth 100 points and will be counted as part of the *Programming Projects* category for the course. Your final score is based on automated tests, as well as a manual review conducted by one of our Teaching Assistants (TAs). A detailed rubric of points and a list of common deductions is below.

Test Submissions

Although you may use any development environment for your project, it must compile and execute on the class server. Make sure it compiles and runs without error on `class-1` before submitting.

You will submit Project 2 to Canvas in the same way you did Project 1. Several optional test submissions will also be accepted. The date and time for test submissions will be announced later. A new `Makefile` is also provided for Project 2.

Emergency Backup

Once you have submitted your project, it is important to keep an electronic copy on a university machine such as the class server (`class-1`) that preserves the modification date and time. You should also make periodic backups or "snapshots" of working versions of your project. If some disaster occurs, it is invaluable to have a working version to which you can revert and regroup.

Grading

This graded assignment is worth 100 points and will be counted as part of the *Programming Projects* category for the course. Your final score is based on common deductions, as well as, a detailed rubric of points.

Grade Standards - Missing: 0%, Poor: up to 50%, Fair: up to 67%, Good: up to 82%, Excellent: up to 99%, Perfect: 100%		
Rubric Points		100.00
1 Compiles on Server		5.00
2 Code Quality		10.00
3 Date class		20.00
4 Vehicle class		25.00
5 VehicleLog class		25.00
6 Driver		15.00
Common Deductions		
Program does not compile ON THE CLASS SERVER (deduction varies depending on how bad, value listed is max)		-60.00
Program compiles but has warnings ON THE CLASS SERVER (deduction varies depending on how bad, value listed is max)		-60.00
Program crashes during execution ON THE CLASS SERVER (deduction varies depending on how bad, value listed is max)		-60.00
Inline function with more than one statement (-10 for each occurrence up to the max deduction listed at the right)		-100.00
Uses abnormal exit (-10 for each occurrence up to the max deduction listed at the right)		-60.00
Uses any global variables		-60.00
Required comments and honor statement not included at start of file exactly as specified		-60.00
Filenames do not follow conventions specified (deduction varies depending, value listed is max)		-60.00
Files missing or added		-100.00
Late penalty for each 15 minutes late		-2.50

Course Materials Notice

The materials used in Georgetown University courses ("Course Materials") generally represent the intellectual property of course instructors which may not be disseminated or reproduced in any form for public distribution (e.g., sale, exchange, etc.) without the written permission of the course instructor. Course Materials include all written or electronic documents and materials, including syllabi, current and past examination questions/answers, and presentations such as lectures, videos, PowerPoints, etc., provided by a course instructor. Course Materials may only be used by students enrolled in the course for academic (course-related) purposes.

Published course readings (book chapters, articles, reports, etc.) available in Canvas are copyrighted material. These works are made available to students through licensed databases or fair use. They are protected by copyright law, and may not be further disseminated or reproduced in any form for distribution (e.g., uploading to websites, sale, exchange, etc.) without permission of the copyright owner.

More information about intellectual property and copyright can be found here:

<https://www.library.georgetown.edu/copyright>.

More information about computer acceptable use policy and intellectual property can be found here:

<https://security.georgetown.edu/it-policies-procedures/computer-systems-aup>