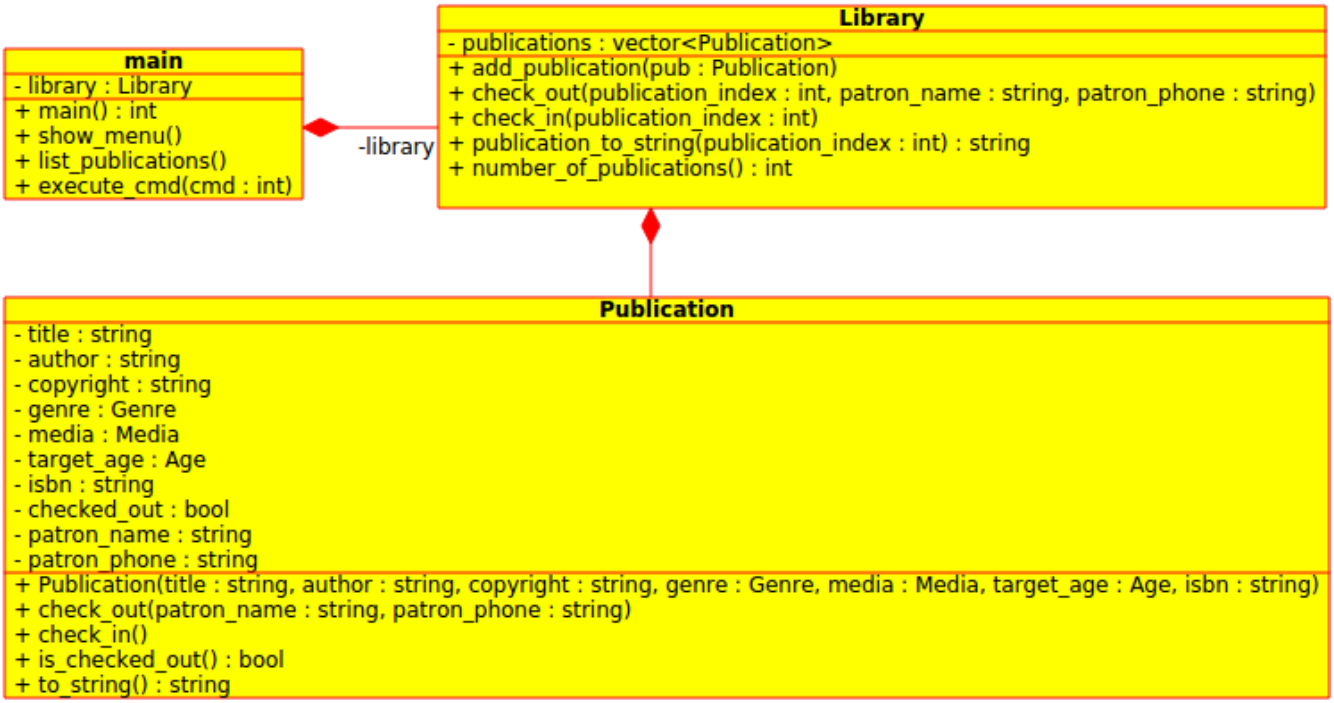
Getting Organized – Library Model

Sprint #1

CSE 1325 – Spring 2018 – Homework #5
Due Thursday, March 1 at 8:00 am

Organizing information is usually left to databases, but object-oriented design principles fit the problem space quite well. In this homework sprint, you'll implement a portion of a simple library management system. (Don't overlook the FAQ at the end!)

Full Credit: Design a Library Management System (LMS) that tracks the **title, author, copyright year, genre, media, target age, and ISBN** for publication in our library. We'll also want to know if the publication is **checked in or checked out** to a customer, and if checked out to a customer, who that customer is (their **name and telephone number**). A possible design is:



You'll deliver **CSE1325_05.zip** with your **updated Scrum spreadsheet** named **Scrum_P5.ods** (or .xlsx or .lnk) a **full_credit subdirectory** containing your **local git repository** including **.h and .cpp files** and a **Makefile.**

Bonus: Same as above, but **break out Patron as a separate class**, list Patrons independent of Publications, and selecting a Patron from a list when checking out a Publication. It is permissible to implement the Bonus directly, without Full Credit, if you like. Same deliverables as above plus an updated **library.xmi** representing your updated Use Case diagram.

Extreme Bonus: Inherit different classes for each media type, managing unique information about each. If you're itching to try your hand at polymorphism, you can do that here, too! Same deliverables as for the Bonus.

Full Credit

The library consists of a lot of publications in several types of media – books, periodicals (also called magazines), newspapers, audio, and video. Each publication falls into a particular genre – fiction, non-fiction, self-help, or performance – and target age – children, teen, adult, or restricted (which means adult only). Each publication also includes a unique ISBN identifier, which is just text.

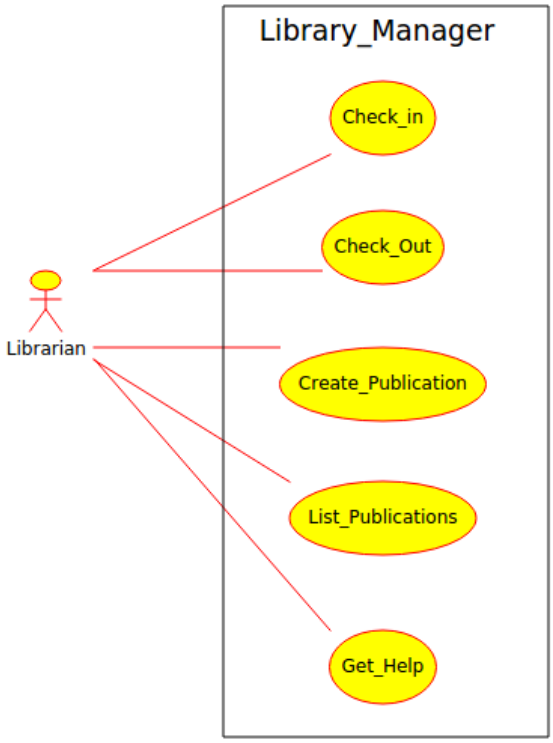
Design an object-oriented application that manages these publications as objects.

We'll want to know the **title, author, copyright year, genre, media, target age, and ISBN** for each of our publications. We'll also want to know if the publication is **checked in or checked out** to a customer, and if checked out to a customer, who that customer is (their **name and telephone number**). It's OK at the full credit level to enter the customer information each time a publication is checked out, but for bonus levels we'll expect to be able to reuse that information from our list of library patrons.

Each publication object should be able to print its contents and its check out status something like this:

> "The Firm" by John Grisham, 1991 (adult fiction book) ISBN: 0440245923
> Checked out to Professor Rice (817-272-3785)

For the first sprint of this multi-week project, we'll target a simple CLI application with 5 operations, as shown in the use case diagram to the right: (1) Create a new publication, (2) List all publications created in the system, (3) Check out a publication to a patron, recording their name and phone number, (4) Check in a publication that was previously checked out, and (5) some short, basic documentation on how to use the system.



(Persistence is NOT required yet. Each time your program is run, it may start without publications.)

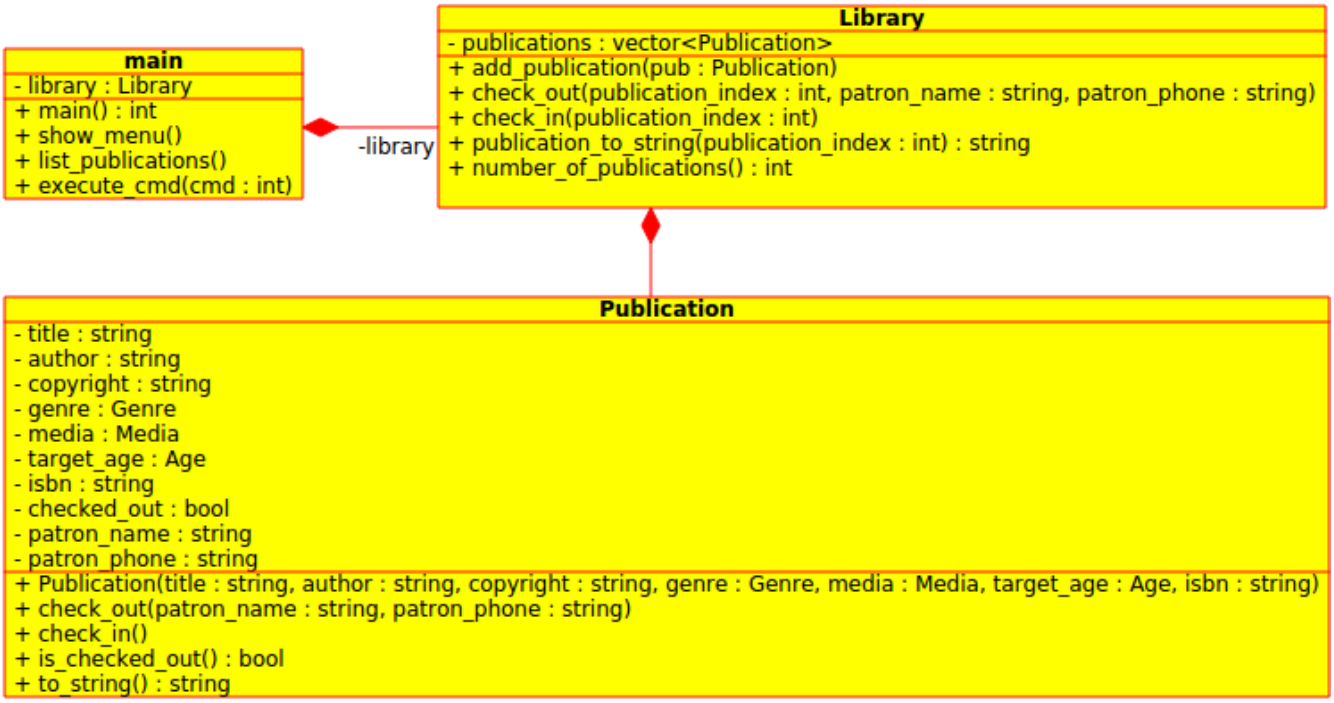
The Use Case diagram for this application is to the right, and the Features for the Product Backlog (in the Scrum spreadsheet) are derived from it below. You should be able to follow how the latter was derived from the former.

(The Scrum_P5.ods file conforms to ISO 26300:2015, and is native to LibreOffice. You should be able to open it in Microsoft Office 2007 SP2 and later directly. You can also open it in Google Sheets by opening Google Drive and selecting New > Google Sheets > Blank Spreadsheet, selecting File > Import..., select the Upload tab in the dialog, drag Scrum_P5.ods from your file manager to the "Drag a File Here" target, and select "Replace Spreadsheet". Other spreadsheets should be able to import the file correctly as well.)

Sprints

Feature ID	Priority	Planned	Status	As a...	I want to...	So that...
AP	1	1		Librarian	Add a publication	I can keep track of what is in the library
LP	2	1		Librarian	List all publications	I can find out what is in the library
CO	3	1		Librarian	Check out a publication	I can track who has borrowed each publication
CI	4	1		Librarian	Check in a publication	I can tell when a publication is returned
HE	5	1		Librarian	Get help	I can learn how to use the system

One possible design is shown in the class diagram and suggested menu below. This is a basic design that meets the requirements above, although more capable designs are likely within your ability now. Therefore, **you are free to create your own improved design**, as long as it meets the requirements. (See the hints in the FAQ for some options to define types Age, Media, and Genre. If you're getting comfortable with C++, also consider attempting the Model-View-Controller pattern, since that may simplify the next two sprints by concentrating user interface into a couple of files.)



Implement the classes as specified (either above or in your own design), **writing appropriate tests for each as you go**, until your system works well.

This is a suggested menu for the Librarian to use with your software. However, **you are free to design your own improved user interface** (including a command line version) as long as it meets the basic requirements.

```
==============================
C1325 Library Management System
==============================

Publications
------------
(1) Add publication
(2) List all publications
(3) Check out publication
(4) Check in publication

Utility
-------
(9) Help
(0) Exit
```

It's possible you'll implement a "secret" option, e.g., 42, that pre-populates some publications to save time when *interactively* testing. Just a gentle hint. If you do, tell the graders – they test, too!

Note that **you will receive partial credit if you only implement a *portion* of the requirements.** You will be graded on the extent to which you cover the requirements and the quality of your code. Be sure to **use header files (.h) for including, and implementation files (.cpp) for implementing** your algorithms. **Use git** to version control your software, and the provided **Scrum spreadsheet** to manage this first sprint. If you don't understand the *intent* of a requirement, feel free to ask – although reasonable assumptions without asking are also fine if you've ever used a library before. If you don't understand the Scrum spreadsheet, ask!!! If you still have trouble with git, ask!!! **If you are unsure about *anything*, ask!!!**

Deliverables

You will deliver to Blackboard a **CSE1325_05 zip** file containing:

- Your **updated Scrum spreadsheet** named **Scrum_P5.ods**, Scrum_P5.xlsx, or Scrum_P5.lnk (containing a link to Google Sheets) for the first sprint *at the root level* – that is, you only need a *single spreadsheet* for managing all of the Full Credit, Bonus, and Extreme Bonus levels.

- A **full_credit subdirectory** containing:
  - Your **local git repository** including **.h and .cpp files** and a **Makefile** that is competent to rebuild only files that have been modified.

  - **Screenshot(s)** as you believe necessary in PNG format to help the graders know what to expect when they "make" (no points are deducted for not having these – your call).

  - (Optional) Your **library.xmi** representing your class diagram only if you didn't use the above design. If you simply implemented the design above, you don't need an XMI file.

Bonus

Modify the design to **add a Patron class** managing the name and phone number of Patrons who check out resources from the library. This adds 2 features to the Product Backlog:

PA	6	1		Librarian	Add a patron	I can conveniently keep track of all patrons
SP	7	1		Librarian	Select a patron when checking out a pub	I don't have to rekey recurring patron info

Add and work off additional tasks to your Sprint 01 Backlog, including the following:

Modify the Library class above to maintain a **vector of Patrons**. Then modify Publication to replace the patron_name and patron_phone fields with a reference to (remember &) a Patron object instead. This will greatly reduce repetitive typing when using the system!

The menu system and some interfaces also need to be modified so that the Librarian can create new patrons (menu item 5) and list all patrons (menu item 6). The dialogs must also allow the Librarian to select an existing Patron from the list of Library patrons during checkout. So your menu *may* look something like this:

```
==============================
C1325 Library Management System
==============================

Publications
------------
(1) Add publication
(2) List all publications
(3) Check out publication
(4) Check in publication
(5) Add patron
(6) List all patrons

Utility
-------
(9) Help
(0) Exit
```

Deliverables

Your Scrum spreadsheet should show completed tasks for the Bonus features. Your **CSE1325_05.zip** file should also contain a **bonus subdirectory** containing:

- Your **local git repository** including **.h and .cpp files** (with tests) and a **Makefile** that is competent to rebuild only files that have been modified,
- Your updated **library.xmi** representing your updated Use Case diagram (you should have 2 more use cases) and class diagram (you should have at least one additional class).
- **Screenshot(s)** as needed to help the graders know what to expect when they "make".

Extreme Bonus

Modify your design from the Bonus level to **use inheritance**, such that each different type of media has a unique field.

- Books – Identify **format** as hardback or paperback.
- Magazines – Identify **publication frequency** as weekly, monthly, or quarterly.
- Newspapers – Identify the **city of publication**.
- Audio – Identify the **format** as CD, cassette tape, or MP3.
- Video – Identify the **format** as Blu-ray, DVD, VHS, or MP4.

This adds 1 feature to the Product Backlog:

CU	8	1		Librarian	Add custom info for each publication type	I can keep better track of library assets

Add additional tasks to your Sprint 01 Backlog, including the following:

Define an output format for each subclass that adds its unique field, and update your user interface, methods, and Makefile to support them.

(Hint: If you choose to use polymorphism, first: **You rock!** Second, remember that in C++ your base class **must** include the “virtual” keyword for every method that will be polymorphically accessed. C++ always makes the journey interesting...)

Deliverables

Your Scrum spreadsheet should show completed tasks for the Extreme Bonus features. Your **CSE1325_05.zip** file should also contain an **extreme_bonus subdirectory** containing:

- Your **local git repository** including **.h and .cpp files** (with tests) and a **Makefile** that is competent to rebuild only files that have been modified,
- Your updated **library.xmi** representing your updated class diagram (your design may vary).
- **Screenshot(s)** demonstrating your programs operation as needed to help the graders know what to expect when they “make”.

Frequently Asked Questions

Q. What does Publication::to_string() do?

As in earlier homework assignments, to_string returns a string containing a textual representation of the object. An example is shown in the requirements. It may be convenient to overload the << operators, but even then it is not unusual for operator<< to delegate the actual string conversion to a class method.

> "The Firm" by John Grisham, 1991 (adult fiction book) ISBN: 0440245923
> Checked out to Professor Rice (817-272-3785)

Q. What types are Genre, Media, and Age?

The types Genre, Media, and Age are left to your preferred implementation. Design something! Here are a few ideas to get you started.

Option #1: They are well-suited to be enums, with the enumerated values given in the requirements:

> The library consists of a lot of publications in several types of **media** – books, periodicals (also called magazines), newspapers, audio, and video. Each publication falls into a particular **genre** – fiction, non-fiction, self-help, or performance – and target **age** – children, teen, adult, or restricted (which means adult only).

The problem with enums is that it's obviously unacceptable to print "age 3" or "media 5" when printing out a publication – you'll need some way to convert a value back to a string.

- Implement a (possibly private) method of Publication to convert the resulting int back into a string for Publication::to_string. Calling these methods "genre_to_string" et. al. may offer a certain consistency, e.g., "`string genre_to_string(Genre genre);`".
- Implement a "helper function" of the same name that exists in global space. This is u-g-l-y from an object-oriented programming perspective, though – we don't leave functions just floating around in global space (it's not even *possible* in Java!).
- Use a vector of strings, e.g.,
  ```
  enum Color {red, green, blue};
  const vector<string> color_to_string = {"red", "green", "blue"};
  ```
  You could convert a Color variable to a string using, e.g.,
  ```
  Color color = Color::red;
  cout << color_to_string[color];
  ```

Similarly, you could define Age with an enum like this:

```
enum Age {children, teen, adult, restricted};
const vector<string> age_to_string = {"children", "teen", "adult", "restricted"};
```

And so on.

Option #2: Or, you can make them each a class with an intrinsic to_string method of their own (called from the publication's to_string method). This is the most object-oriented solution, but you are NOT required to do this! If you do, Age might look something like this:

```
#ifndef __AGE_H
#define __AGE_H

#include <string>
using namespace std;

class Age {
 public:
  Age(int val) : value(val) { }

  static const int children = 0;
  static const int teen = 1;
  static const int adult = 2;
  static const int restricted = 3;

  static const int num_ages = 4;

  string to_string() {
   switch(value) {
    case(children):return "children";
    case(teen):return "teen";
    case(adult):return "adult";
    case(restricted):return "restricted";
    default: return "UNKNOWN";
   }
  }
 private:
  int value;
};
#endif
```

Option #3: Or, to stretch the definition a bit, you could just make them a string, either directly or via typedef, which we haven't covered yet - see http://www.cplusplus.com/doc/tutorial/other_data_types/. The downside is that you'll need some significant data validation code to ensure that *every string in every object is always correct*. If you just let users type whatever they want, you'll have a disaster instead of a database.

Q. The UML shows a Main class with a main() method. How is this implemented in C++?

Remember that the **UML is not specific to C++** - a UML class diagram can be implemented in any language. Since some languages (looking at you, Java) require that main() be part of a class, a UML class diagram *may* show the design in this way

As discussed in previous assignments, however, C++ is unable to specify main() as part of a class, so it **must** be implemented as a stand-alone function, e.g., int main( ) { }.

Part of understanding UML is to be able to translate from constructs valid in the UML to constructs that a C++ compiler can handle. This is one such case.

Q. What type do I return if the UML specification of a method doesn't list a type at all?

We always implement the return type of a method specified in the UML with no return type as a void in C++.

For a constructor, which of course is not a method, we would never specify a return type in C++ - the return type is an object, and is implicit.

Q. How do I implement main.h?

Since main.cpp just contains the main() function, which is invoked when you run the program, you don’t need and shouldn’t create a main.h file.

Q. How do I deal with check_in and check_out methods in both the Publication and Library classes? Don't the conflict with each other?

No. An object instanced from the Publication class represents an individual book, magazine, DVD, etc. Think of one of the textbooks for this class. When you call check_out on this object, you are checking out the associated media - thus you only need to know the patron_name and patron_phone of the person who will temporarily take possession of the associated artifact.

An object instanced from the Library class represents a *collection* of books, magazines, DVDs, etc. Think of the UTA library, which contains a large collection of media. When you call check_out on this object, you must also specify which specific publication to check out from the library.

Similarly, calling check_in on a publication object requires no parameters - it's checking in the publication associated with that object. But calling check_in on the library object requires that you specify as a parameter which publication in the library is being checked back in.

If this worries you, consider using different names for each class’ methods. That’s not necessary, but it’s certainly permissible.