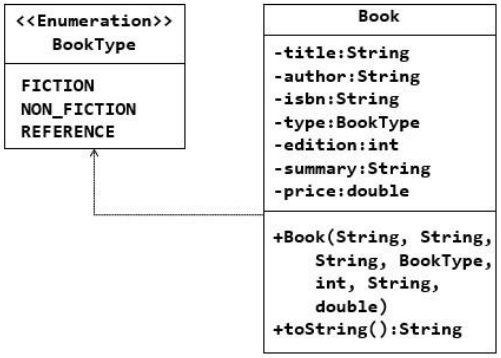
A2 Guidance – Making a Start
(Some Useful Activities for Reading Week)

1. Class Book



1. Start with class **Book** and enum **BookType**
2. Add the **instance data** for class **Book**
3. Add the **constructor** for **Book**
4. Add the **toString** method for **Book**
5. Add what you think would be appropriate **getters** & **setters**.
6. Remember – read the information on class **Book** and observe stated constraints.
7. Create a **Test app** – and test instances of class **Book**.

2. QUBLibrary Application

This is the main application class for **Part 1**.

You may use the **Menu** class (as described in the lectures) – you could also consider updating this or proving another of your own design – but for **Part 1** - it must work with the standard Eclipse output console (and Scanner) for input.

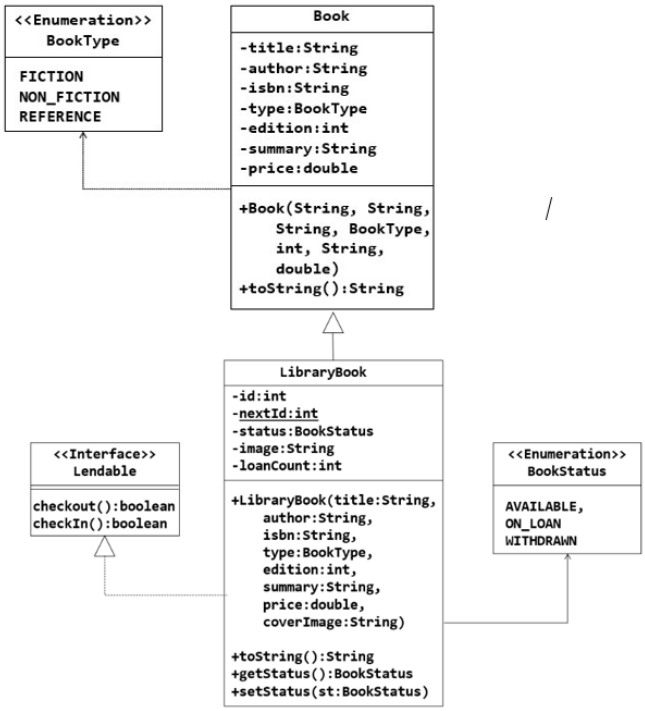
You could begin with a **basic application**, using class **Menu** to provide the specified options for **Part 1**. As an outline application, on selecting an option the initial response could be just a message displayed on the console. This would give you the basic application structure for further development.

3. Class LibraryBook

Development of this class will require a little information on a topic call **Inheritance** – which we will look at **next Tuesday**. However, feel free to look ahead and read up a little on this topic.

A2 Guidance – Time for Some Inheritance!

1. Class LibraryBook



1. Use **inheritance** to define class **LibraryBook** – include the specified attributes
2. Add in a **constructor** for this class
3. Remember to call the superclass constructor (using **super**)
4. Add the required methods – **override** 'toString'
5. This class makes use of an **interface** called 'Lendable'.

You easily create the 'Lendable' interface:

1. Use the Eclipse menu to add a new **interface** – call it 'Lendable' - it should resemble the following:

```
public interface Lendable {
	public boolean checkout();
	public boolean checkin();

}
```

2. The 'Lendable' interface is just a list of method headings (as you can see above).
3. The interface is used in the definition of class **LibraryBook**, which should look like the following:

```
public class LibraryBook extends Book implements Lendable {
	// rest of your code
}
```

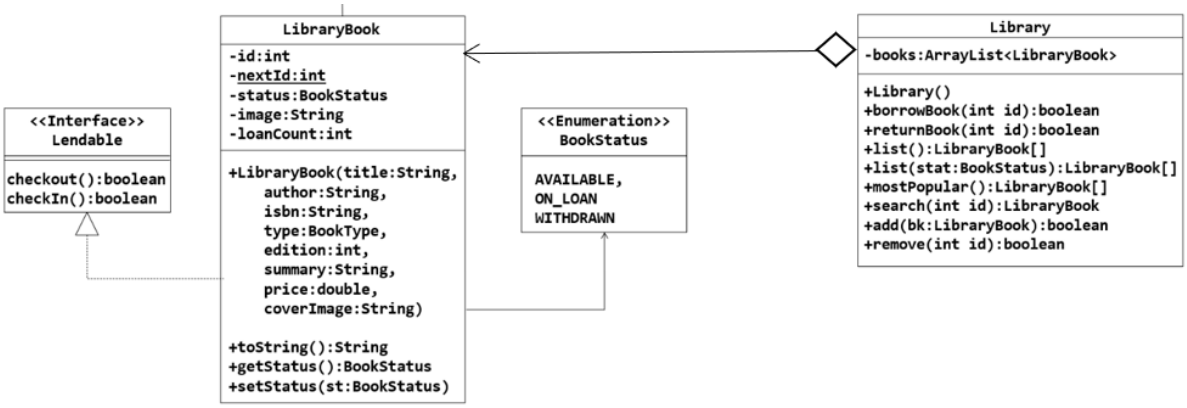
When you use an interface (as above) you are telling Eclipse to make sure that the class includes definitions for all methods in the interface. In other words, class **LibraryBook** must contain the following methods:

```
public boolean checkout() {
	// attempt to checkout
	// (borrow) a book
}
```

```
public boolean checkin() {
	// attempt to checkin
	// (return) a book
}
```

2. Class Library

This class manages a collection (**ArrayList**) of **LibraryBook** instances.



Begin by providing **outline** implementations for **all required methods** – for example:

```
public boolean borrowBook(int id) {
      return true;      // you can then complete each in turn (then test)
}
```

3. Class QUBLibrary (Application class)

You could consider using the 'Dog' application from the practical material as a basis for the structure of your application:

- Put the **menu** in place (with the required options)
- Provide a separate **method** to manage each possible option – these methods could simply print a message to the standard console (to say that it's been called)
- Each of these methods could then be implemented in turn to add the required behaviour.

INTRODUCTION

In your role as a software developer, you have been contracted by QUB's Library Department to create a Java application capable of managing data on library books. Specifically, you are required to build and test a prototype system that meets the deliverables outlined in the sections below.

GENERAL INSTRUCTIONS - CODE IMPLEMENTATION

1. Create a new Java project named 'Assessment2'.
2. Add three packages to 'Assessment2' named part01 and part02.
3. Implement your code as described below:
    - 'part01' should contain all code associated with the core requirements.
    - 'part02' should contain the code related to implementation of additional features only, including the use of class Console, to be introduced in the material for Practical 4.

SUBMISSION DATE/TIME – FRIDAY 14TH MARCH 2025 BY 5 PM.

PART 1: CORE FUNCTIONALITY (60 MARKS AVAILABLE)

To demonstrate that the proposed Java application is feasible, the following must be implemented.

1.1 Implementation of the classes/enums/interface shown in Figure 1 below.

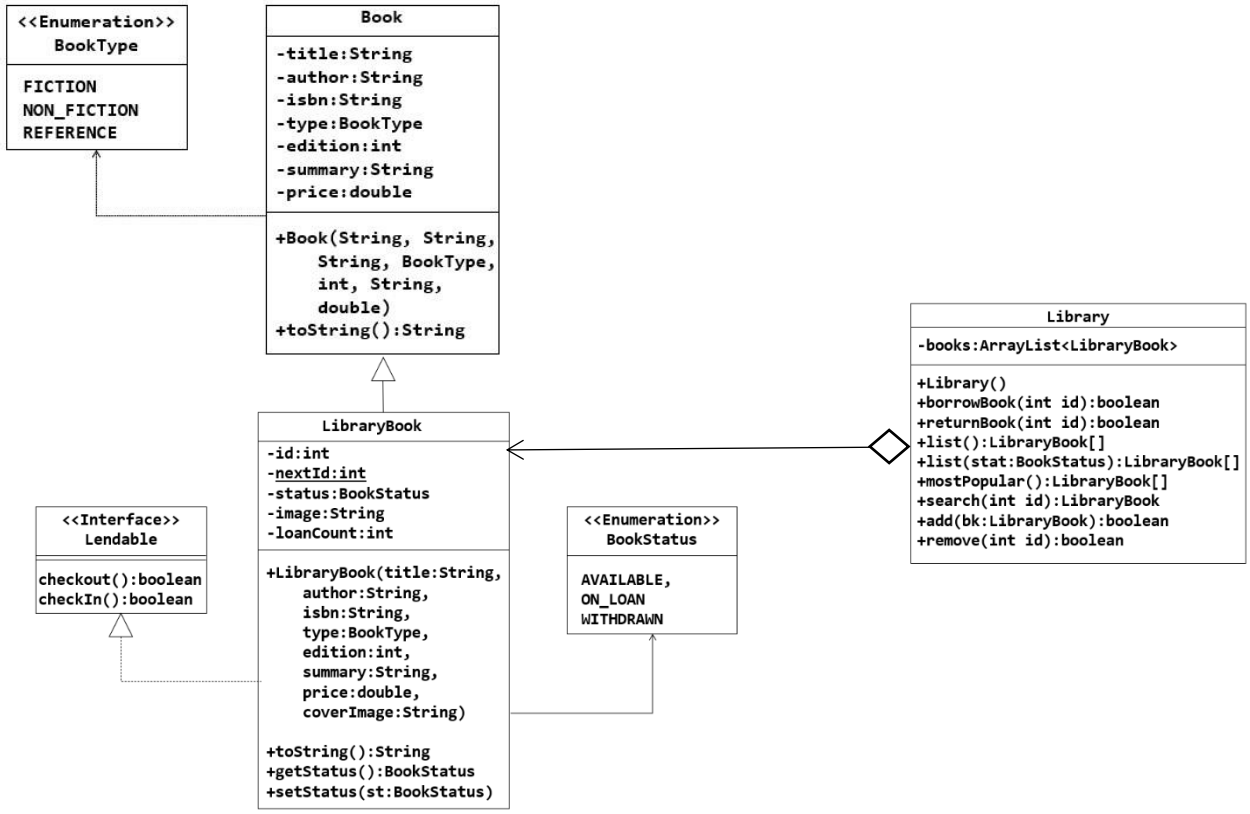


Figure 1: A (partial) UML Class Diagram for QUB Library Management Software.

Further details on the role and behaviour of the above classes follows – ***please read the entire document before starting***.

CLASS INFORMATION

Book Attributes and Methods:

title (String) – the book title (minimum 5, maximum 40 characters)
author (String) – the book author (minimum 5, maximum 40 characters)
isbn (String) – a String of exactly 10 characters (each must be a digit 0..9)
type (Booktype) – the type of book
edition (int) – the book edition (must be greater than or equal to 1)
summary (String) – a summary of the book contents (minimum 20, maximum 150 characters)
price (double) – the cost of the book in £
Book constructor: parameters in order – title, author, isbn, type, edition, summary, price)
toString – returns a String detailing all book attributes (on a single line)

LibraryBook Attributes and Methods:

id (int) – a unique identifier for a LibraryBook instance.
nextId (int) – next usable identifier, to be used when assigning **id** above.
status (BookStatus) – determines the status of a book (available to borrow, on loan, book withdrawn).
image (String) – the filename for the book cover image (without path).
loanCount (int) – number of times the book has been borrowed.
Constructor – parameters (in order): title, author, isbn, type, edition, summary, price, coverImage.
toString – returns a String detailing all library book attributes (on a single line)
getStatus – returns status of the book.
setStatus – sets the status of the book.

Library Attributes and Methods:

books (an ArrayList) **–** contains references to all books in the system.
Constructor – initialises a Library instance.
borrowBook – a request to borrow a book identified by id.
returnBook – a request to return a borrowed book (identified by id).
list (no parameters) – returns an array of all library book instances.
list (status) – as above but includes only books which match the supplied status.
mostPopular – returns an array of LibraryBook instances ordered by number of times borrowed.
search – returns a LibraryBook reference for id parameter or null if it does not exist.
remove – removes a LibraryBook (identified by id) from the books ArrayList – returns boolean to indicate success.
add – collects data for and adds a LibraryBook instance to the books ArrayList, if the following criteria are met:

i) Constraints for Book attributes are applied
ii) price must be greater than £0.00

1.2 A standard text console (menu-based) application (called ***QUBLibrary***) to demonstrate (through a ***Library*** instance) the interaction with ***LibraryBook*** instances within the system. The following menu options should be provided:

- ***List All Books***: the user should be able to view details of all books, one set of book details per line/row of output.
- ***List Books by Status***: as above but only showing the details which match the status selected by the user.
- ***Add a Book***: user should be able to request book information from the user, adding a new book to the system.
- ***Remove a Book***: a user should be able to select and remove a book from the system, marking it as 'withdrawn', if it is not currently on loan.
- ***Borrow a Book***: a user should be able to select from a list of books available to borrow (and borrow it).
- ***Return a Book***: a user should be able to select from a list of books on loan (and return it).
- ***Display Ranked List***: a user should be able to view book summary information (title, author, number of times borrowed), in order of the most popular books.
- ***Exit***: the user should be able to exit the system.

Notes:

- **A.** The application should create a ***Library*** instance.
- **B.** By default, a minimum of 5 ***LibraryBook*** instances should be available to view/manage in your system on startup – do NOT use file storage for this.
- **C.** Care should be taken over input/output layout using messages to keep the user appropriately informed.
- **D.** You should make use of the ***Menu*** class defined in Menu.java (from the assessment specification on Canvas).
- **E.** Do not add additional public methods to class ***Library***. You may add getter/setter methods to ***Book*** or ***LibraryBook*** as appropriate.
- **F.** When using ***ArrayList***, you are restricted to the ***get***, ***add***, ***size*** and ***remove*** methods only. Searching or sorting algorithms must be coded in line with the material covered in the lectures.
- **G.** As '**LibraryBook'** makes use of interface 'Lendable' – you will need to provide implementations for listed methods:
  - **checkout**: to request that a book be marked as 'borrowed'.
  - **checkIn**: to request that a book be marked as 'returned'.

PART 2: A CONSOLE EDITION (40 MARKS)

3.1 Using the ***Console*** class (to be introduced in Practical 4), implement a new application called ***QUBLibraryUpdated*** which replicates the behaviour of ***QUBLibrary*** from ***Part 1***. Specifically, the application should:

- Make use of ***Console*** instance(s) for all input and display functionality.
- Use images, fonts and colour as appropriate to enhance the user experience – add a standard folder called 'Images' to the project – to contain all images files.
- Apply the same coding restrictions outlined for Part 1 above, aside from using **CSC1029Console.jar** containing the definition of class **Console.** Do NOT make use of external files other than those provided in your 'Images' folder.

Marks will be allocated for functionality/behaviour and creativity in the use of the ***Console*** class. This class and its behaviour will be discussed in forthcoming lectures and lab sessions.

Use of the ***Console*** class will require adding an external library (jar) to your project. Instructions on how to use this file are included in the appendix.

CHAT GPT (OR OTHER AI TOOL)

You are required to provide a brief report indicating where and how you have used AI in your work for this assessment. Include the following information for part 1 and 2:

1. Details of Chat GPT (or other software) used,
2. Details of use (e.g. for code generation, debugging)

If have not used Chat GPT in any way – please state this in your (very brief) report.

Note: You may freely use AI for debugging assistance but NOT for code generation. You must code the solution yourself**. You are restricted to using the Java language features detailed in the material covered at the start of the module. Other (more advanced) language features are not permitted.**

SUBMISSION INSTRUCTIONS

1. Create a new folder to hold all your assignment files for submission. The folder should be suitably named and include your student number.
2. Copy and paste the 'part01' and 'part02' folders from your java project into the folder created in step 1. ENSURE that these folders contain all of the corresponding .java files.
3. Include your 'Images' folder – containing a maximum of 10 book images
4. Include your report indicating use of ChatGPT.

5. Verify that you have all the required files included in the folder created in step1.
6. Compress the folder to a zip file and upload it to the assignment location in the CSC1029 module on Canvas by **5.00pm** on **Friday 14th March 2025.**

NOTE: Please check that you have included the original development Java (.java) files. If you submit .class files these will NOT be marked. These submissions will be date-stamped and in accordance with University regulations, late submissions will be penalised. **Failure to follow the submission instructions may result in a reduced or zero mark for part or all of the assignment!**

APPENDIX 1: HOW TO INSTALL AND TEST THE EXTERNAL LIBRARY FILE (JAR)

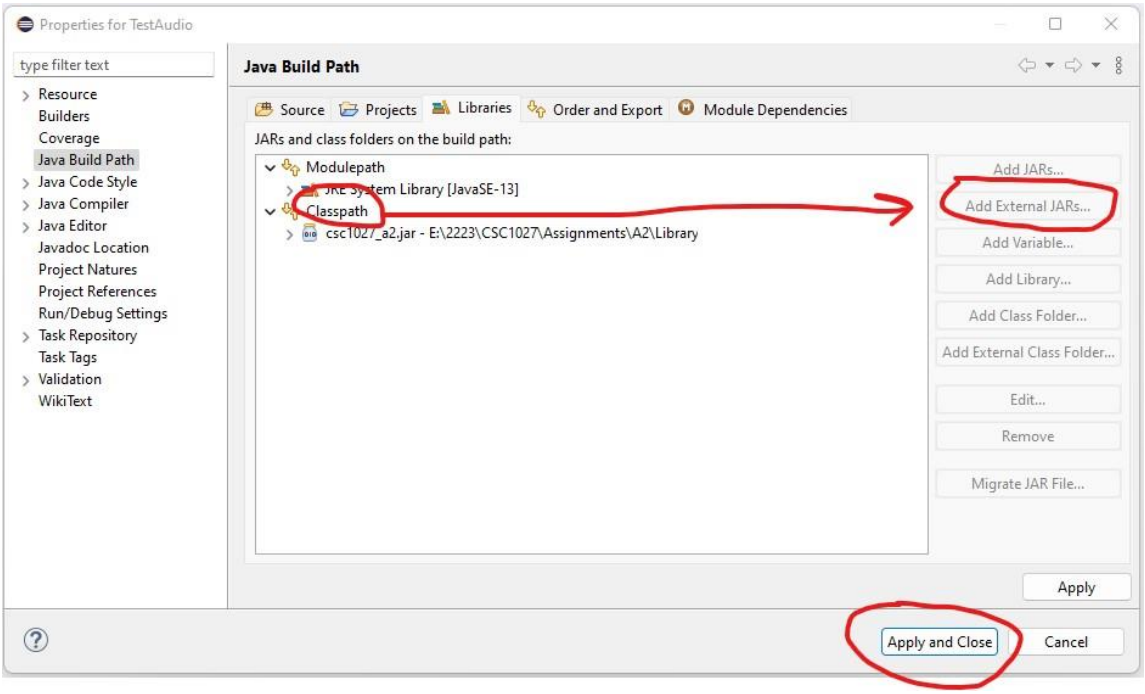
Step 1 – Download the following files from Canvas (under Assessment 2)

- ***csc1029_a2.jar*** – this file contains the definition for class ***Console***

Step 2

Add the downloaded jar file to your project:

- From the package explorer in Eclipse, "***right-click***" your project directory (***TestAudio***) and select "***Build Path->Configure Build Path ..***"
- Select the "***Libraries***" tab and highlight "***Classpath***"
- Click the "***Add External Jar***" button (on the right-hand side) and navigate to the downloaded Jar file to select/open.
- Click on the "***Apply and Close***" button



<-- Your Package Explorer should look something like this

<-- The Jar file should be visible within "Referenced Libraries"

Note: Do not use module-info.java in your project configuration – please make sure this is removed.:

APPENDIX 2: ASSESSMENT CRITERIA

Part 1 – 60 %

This part of the assessment requires code implementation based on a UML specification and is assessed under the following criteria:

- ***Definition Accuracy*** – code definitions (class name, attributes, methods, comments) must match the UML specification
- ***Behaviour/Functionality*** – on execution, code must function as described in the specification. Your application should be robust and should carefully manage input/output to provide an efficient user experience, within the limitations of the standard Eclipse console.

Part 2 – 40 %

This part of the assessment requires implementation of some additional functionality which uses the definitions/code created in Part 1. Assessment criteria:

- ***Correct definitions*** within naming conventions
- ***Correct behaviour/functionality*** as per the specification
- ***Accuracy/Usability/Code Quality/Commenting***
- ***Creativity/Innovation in using the Console class***