Overview **Given out Wednesday, Week 5**

Your code from Part 1 should successfully read in the data for a library item from the text file called `item_data_1.txt`. In this part of the project, you will make it easier to deal with different types of library items by introducing subclasses of the `LibraryItem` class and using inheritance to enhance the usability of your code.

In this project, you will be given
- more than sufficient advice on how to code this project including advice on potential errors;
- more detailed help in parts of the project that we think you may find more difficult than other parts of the project;
- **But** you will not be given 100% instructions at every step *nor continually reminded* of potential pitfalls. Learning to debug your own code is a skill honed by practice.

Reading Tip

Before starting to code, you will find it useful to have another look at the slides relating to the `Network` project of Chapter 10 and note that:
- The object `post` had static type `Post` but its dynamic type – its type at runtime - was either `MessagePost` or `PhotoPost`.
- It was this dynamic type which determined which `display()` method was executed. For example, if the dynamic type was `MessagePost` then the `display()` method of the `MessagePost` class would be executed. This is called method overriding.
- Since the `display()` method of the sublass would be executed in the above example, in order to output information contained in the superclass's fields then that subclass' `display()` method needed to call its superclass's `display()` method.
  - o Note also that this superclass call could be written either before or after the subclass's field details were printed out.
- The above notes (referring the `Network` class and the `display()` method) also apply to any similar method which is in both a subclass & a superclass ... such as `readData()`. Hint, hint.

Step 1 Overview in more detail

In this step, you will use two new data files: `item_data_1_v2.txt` (note this is a `v2` of the original file) and `item_data_2.txt` contain more library item data. Look at each of these data files.

The data file `item_data_1_v2.txt` contains more data about a book than before as the client has now asked for more data. In particular, *in addition to the previously seen data* of `title`, `itemCode`, `cost`, `timesBorrowed` and `onLoan`, the book data now includes data for the author of that book, its ISBN number (an international code identifying each book), the number of pages in that book and the publisher of that book.

If you examine the second data file, you will see that the data here has been split into two types: book data and periodical data (where a periodical is a magazine or newspaper published at regular intervals). The book data is the same as in `item_data_1_v2.txt` and the data for periodicals as well as having the book details above has the additional data of its publication date in the format of e.g. 05-02-24.

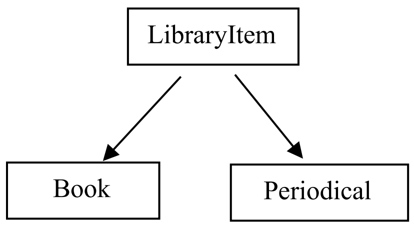
This file also contains lines such as

```
[Book data]
```

that will not be relevant until later though it should be obvious that these are labels or flags that signpost the type of data immediately below this label.

Before starting to code, read carefully all of this step.

The library offers a variety of items to its borrowers and these items are organised as shown in the diagram below :



Book and **Periodical** are direct subclasses of **LibraryItem** and, to be clear, as well as the fields inherited from the **LibraryItem** class ….

- the **Book** class has two additional fields: **author** and **isbn**;
- the **Periodical** class has one additional field: **publicationDate.**

What we will next do is to write code that

- firstly creates the **Book** class and links it with the **LibraryItem** class as shown in the diagram above;
- then fills the contents of the fields in both **Book** and **LibraryItem** with data from **item_data_1_v2.txt;**
- then creates the **Periodical** class, links it with the **LibraryItem** class as shown in the diagram above; and fills the contents of the fields in both **Periodical** and **LibraryItem** with data from **item_data_2.txt** (as well as filling the fields in **Book** with appropriate content).

Once you understand the above notes … carry on reading below .. but do NOT START CODING yet.

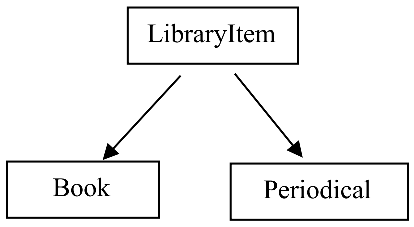
As detailed above, let's consider firstly reading the data for books. It is obvious that, in the **Library** class, we could write a method **readBookData()** - that would be similar to a method **readPeriodicalData**. But, as our model grows, this approach means that we will end up with many "read" methods in this class e.g. **readBookData()**, **readPeriodicalData()**, **readAnyOtherSimilarData()** etc. If we also decide to separate our data into separate classes for each type then we will also end up with many different data files. ☹ A more sensible approach is to have one "read" method, as we have now, in the **Library** class and also to keep all the data in **one data file**. This is the approach that we will adopt here.

Remember that in the last step of Part 1, we let a **LibraryItem** object ***read its own data***. Now, as a **Book** object holds the data specific to books and a **Periodical** object holds the data specific to periodicals, it makes sense for the same idea to be used with these classes i.e. the **readItemData()** method lets the data for a **Book** object be read by the **Book** class and similarly for the data for a **Periodical** object to be read by the **Periodical** class.

If you understood the notes above, then you will realise that – because you are going to use read methods in each class to populate your fields – then your constructor *with parameters* in **LibraryItem** is no longer needed and should be deleted. Also, any default constructor that you have in **LibraryItem** is no longer needed for similar reasons and should now be given an empty body

Step 2 Starting coding

We shall now add our first subclass: **Book**. Here again is our simple structure,



Before you start coding, consider each piece of data in both files and decide into which class (`LibraryItem` or `Book` or `Periodical`) that data belongs. If this is not clear then go and re-read the advice above. Or ask a tutor.

Now start coding by

- add our first subclass, `Book` which should have the fields mentioned above;
- introduce a `readItemData()` method in the `Book` class. This should be the method that is used at run-time to read the data into this object.

At this point, your project should still compile and should read the data corresponding to the extra fields of the `Book` class. So, test your code by reading data from the text file, `item_data_1_v2.txt`.

If your code causes an error then the most likely one is a `NoSuchElementException` error. This means that the scanner has met a piece of data of a different type than it expected to meet. Try to discover the error by looking at this data file to see what is new that could cause this exception before reading the next paragraph.

The most likely reasons for this new `NoSuchElementException` error are as follows:

- Did you remember to change the type of object created by the read method in `LibraryItem` ? After all, `LibraryItem` is a superclass and you should not create such an object.
- The code that is executing is e.g. `scanner.nextInt()` and the next piece of data is a string;
  - To check which data caused the error, add a `println()` statement to your read method to print out which data was being read in at the time of the error, and then debug accordingly.
- To make sure that method overriding occurs (see again the **Tip** above) then you must make sure that all read methods have the same name.

Once you can run your code without getting that error then this is indeed progress ! 😊 But **do not move onto the next part** without getting all errors sorted out !

Remember that we can either only check that the read methods have correctly worked by using the Object Inspector or by calling the print method which prints out each field's contents – see next bullet.

- Now introduce a `printDetails()` method in the `Book` class that overrides the `Library` class's `printDetails()` method, in a similar manner to that employed by the `print()` method of the `Network` project;

Finally, for this step, a little bit of refactoring: because we are delegating the reading of data to the subclasses, we no longer need an inner while loop `in readItemData()` (if you did have one that is ☺), and this can safely be removed now.

However, you might like to think whether the check (for your remaining `while` loop) that you have is the most appropriate.

Step 3 Dealing with data other than book data

We are now ready to attempt to read from a text file that contains data for both books and periodicals. The file will be in a similar format to `item_data_2.txt and` will be structured like this:

```
[Book data]
data for a book in the same format as item_data_1_v2.txt
data for another book
...

[Periodical data]
data for a periodical in the format
      publicationDate, noOfPages, publisher, title, itemCode, cost, timesBorrowed,
      onLoan
data for another periodical
...
```

i.e. each block of data for a particular type of item is preceded by a label (called a flag) (such as `[Book data]`) so that we know what kind of data to expect in the lines of the text file which follow the flag. You can assume that any line that starts with a `[`, ignoring any leading spaces, is a flag indicating a type of data.

So, each time you meet a flag, you need to record in your code what type of data comes next in the file. Why ? Ask a tutor if unsure. Use a local variable named **`typeOfData`** for this purpose.

Now your code may be a little different, but your read method in **`Library`** should have a loop that looks something like this (if not ☹, seek help from a tutor):

```
while (there are more lines in the data file )
{
   lineOfText = next line from scanner
   if ( this line starts with // )
      { ignore this line}
   else if (this line is blank )
      { ignore this line}
   else
      { code to deal with a line of Book data }
}
```

*In the following, the basic loop structure above will **not** change -- in particular, data should be read from the file **only** at the beginning of the loop.*

The new pseudocode for the loop above should now be altered to look like this:

```
while (there are more lines in the data file )
{
   lineOfText = next line from scanner
   if ( this line starts with // )
      { ignore }
   else if ( this line is blank )
      { ignore }
   else if ( this line is a flag )
      { set typeOfData }
   else
      { deal with the data appropriately }
}
```

Other hints:

- The file is essentially "comma separated" rather than "space separated" and it may also contain spaces before or after a comma. So, if you have not already done so, amend your call to **`useDelimiter()`** so that it uses a regular expression that represents zero or more spaces, followed by a comma, followed by zero or more spaces.
- At this stage, the data for periodical objects cannot be read in because we have nowhere to store that data. Hence, you need to write code that ignores this data.

Step 4

Next add the **`Periodical`** class (with the relationships shown in the diagram on Page 2) to the project and amend the read method in **`Library`** so that it recognises this new data. If you have successfully completed the previous steps with suitably structured code then this should be straightforward although, as usual, you will need to pay attention to detail. In fact, your **`Test`** class remains the same for this step as for the last few previous steps ☺.

Although, at this stage, you could simply deal with book and periodical data, better marks will be awarded for dealing with other possibilities i.e. for "unexpected flags". If such a flag is encountered when reading from the file, print out a warning message though you will have to add a little more code than a single println(). ☺

Step 5

You should have noticed that we never created instances of the class **`LibraryItem`**. It is similar to the class **`Animal`** in the foxes-and-rabbits project in Chapter 12 and if you have not already done so, now make **`LibraryItem`** an abstract class.

And finally, have a check over your code to make sure that you don't need to refactor it to make it even better ☺

In addition, are you still storing the data files with your Java files ? If so, put them in a more appropriate place (and amend your code accordingly).

You should aim to complete this part of the project (or as much of it as you can) by 28th February when Part 3 of the project will be made available.

To try to keep you to a good schedule, you must upload the work that you have completed to date – that is, Part 1 and as much of Part 2 as you have completed - by 16.00 Wednesday 28th February at the latest.

If you do not upload your work to date within 24 hours of this deadline then we will not mark the project (at the end of the semester).

Part 3 of the project will be made available that same day.