**COM1003 Problem Sheet**

**This is an individual piece of assessment; you are not allowed to work in groups and your code must be all your own work. You may not use other peoples’ code or ideas, e.g., taken from the Internet using ChatGPT. Any form of unfair means (plagiarism or collusion) will be treated as a serious academic offence and will be processed under our University Discipline Regulations. These might result in a loss of marks, a possible failure of the module or even expulsion from our University. Read further details in the** [**UG Student Handbook**](https://sites.google.com/sheffield.ac.uk/comughandbook/your-study/assessment/unfair-means)**.**

**Before you begin, you should have completed the Online Quiz 1 and the lab sheets from weeks 1-4. To complete this problem sheet you will also need to cover the material for week 5.**

**You may ask demonstrators for help in understanding the problem, but as this is intended to be an individual assignment, they will not be able to help you directly with programming tasks.**

**This is a live problem sheet: you are welcome to add questions or suggestions to it as** [**comments**](https://support.google.com/docs/answer/65129?hl=en-GB&co=GENIE.Platform%3DDesktop#zippy=%2Cadd-a-comment)**, which will be addressed by your instructors. Keep an eye out for highlighted excerpts, because these are corrections/improvements made to the problem sheet as a result of your comments.**

**Following the submission deadline, this worksheet will no longer accept comments / suggestions.**

# Introduction (video brief to come)

Similarly to how you downloaded the code for the second semester ([see Lab1](https://docs.google.com/document/u/0/d/1K6UgY9gGkiPS2UhOnzwCA4j-MaG2z5SS/edit)), you are now provided with a Gradle project that implements a simple library system. The code of the project is organised in a main package named uk.ac.sheffield.com1003.library which is split across the following directories:

* src/main/java contains the source code of the project, including the classes that you should be implementing or completing.
* src/test/java contains JUnit tests which you can execute to assess if your solution is working as expected. You should **not** modify any of the classes in this directory. If you do, it will not be possible to assess your submission and you will obtain a **zero mark**.

| To accept the assignment, visit this [GitHub Classroom Assignment](https://classroom.github.com/a/h8Eu91FT). **It is important that you pick your own student username to accept the assignment (it will be linked to your personal GitHub account).** If you have correctly linked your student username (e.g., aca1jrs) with your Github account for the labs repo, then you will not need to link them again. If you cannot find your own student username in the list or accidentally link your Github account to another student’s username, email j.rojas@sheffield.ac.uk. |
| --- |

In the source directory (src/main/java) of the provided project, you will find the following classes:

* **Library**: A library has a **name** of type string, a private array **CatalogueItem[] catalogue** of reading items, e.g., books, a private array of loans (i.e., **Loan[] loans**), and a **loanLength** field which specifies how many days books are loaned for.
* **Loan**: Represents an item being borrowed by a user from the library. The information contained in a loan includes the item being borrowed, the user (instance of class **Person**) borrowing the item, **loanDate** which is the date and time when the loan was made, **dueDate**, which is set to the date the loan is created plus the library’s specific loan length, and the date the book was returned (**returnedDate**, initially set to **null**).
* **CatalogueItem**: Represents a reading item in the library catalogue. It contains the following fields: a **title**, the publication **year**, and the number of **copies** owned by the library.
* **Book**: Extends class **CatalogueItem** with the following fields: **author**, of type **Person**, **isbn** and a **genre**, which can be one of **UNSPECIFIED**, **FICTION**, **NONFICTION**, or **NOVEL**.
* **Person**: Represented by a **firstName** and a **lastName**. Authors of books and users of the library are **Person** instances.

The **uk.ac.sheffield.com1003.library.TestLibrary** class in **src/test/java** contains some tests that you may find helpful to develop a correct solution.

***Note****: You may notice that some of the classes present some level of redundancy or inefficiency that we would ideally like to avoid (e.g. by using public abstract classes and/or interfaces). This is intended to avoid unnecessary complexity and keep the problem sheet achievable.* ***Refrain from making changes to the provided code (in particular class and method signatures) and focus on implementing the required functionality as requested in this problem sheet****.*

# Tasks

By browsing the project’s source code, you will notice that most of the public methods in the classes mentioned above are either missing or incomplete. Your goal is to implement **all** these methods according to the instructions provided in the methods’ Javadoc. These are your tasks:

1. As you will notice, your project is missing a starting class. Create a new class named **App** in package **uk.ac.sheffield.com1003.library** with a **main** method in which you create an instance of type **Library** with name “Sheffield Central Library”, 10 days as loan length, and the following two **Book** instances added in the **catalogue**:
	* **Book 1**: **“Clean Code”**, written by **Robert Martin**, published in **2008** with ISBN **9780136083238**. For this, you will have to implement constructors without parameters (i.e., *default constructors*) for the classes **Person** and **Book**.
	* **Book 2**: The book represented by the Bibtex entry contained in file **src/main/resources/Lee60.bib**. You will need to read the content of the file in the **App.main** method into a string variable and then call the static method **Book.fromBibtex(String bibtex)** to create an instance of class **Book**. You may assume that the format of the Bibtex entry will be as follows:

@book{<an alphanumeric key>,
 title = {<A title made of any characters except curly braces>},
 author = {<Author’s last name>, <Author’s first name>},
 isbn = {<Exactly 13 digits, starting with prefix 978 or 979>},
 year = {<valid year, 4 digits>}
}

You are only expected to extract these four fields, not all the [fields supported in a bibtex entry](https://www.bibtex.com/g/bibtex-format/). You should ignore the alphanumeric key in the first line between “@book{” and the first comma. Commas separate fields, and there will not be a comma after the last field. The title and author names will not contain additional quotes. There might be any amount of blank space between fields and around the ‘=’ character.

 For this, you will need to implement and use the method **Library.addItem(...)**.

1. Implement method **Library.printCatalogue()** and call it from method **App.main**. You will have to implement Library.welcome() and call it from **Library.printCatalogue()**. See the method’s Javadoc for guidance. The test TestLibrary.testPrintCatalogue() might be helpful, too.
2. The library is now ready to start loaning books, but you need to implement methods **Library.loanItem(...)**, **Library.returnItem(...)** and **Library.extendLoan(...)**.
	* The method **loanItem** may throw two types of exception: ItemNotFoundException (if the item received as argument does not exist in the catalogue) and NoCopyAvailableException (if all the available copies for a given book are already on loan). You must create these exception classes and use them appropriately (e.g., by declaring that the method throws certain exception classes).
	* The methods **returnItem** and **extendLoan** may throw **ItemAlreadyReturnedException** if the item has already been returned, i.e., **returnedDate** is not null.

 Extend **App.main** by creating two loans in the library (calls to **Library.loanItem**), returning one of the items (call to **Library.returnItem**) and extending the other loan by **5** days (call to **Library.extendLoan**).

1. Now that the library is up and running, the staff would like you to implement a method that will print the list of books that are on loan and overdue, i.e., **Library.printOverdue()**. The Javadoc for that method will tell you the format in which the list should be printed. Add a call to printOverdue at the end of **App.main**.
2. As the library grows, multiple editions of the same book are added. When a user requests a book and all the copies of that particular edition are loaned, the librarians wish to be able to offer a different edition of the same book which might be available. To help with this, you must implement method **Book.equals(Book other)** to decide if two book instances correspond to the same title. Two books should be *equal* if and only if:
	* The title of one is a prefix of the title of the other, e.g., “On the Origin of Species” and “On the Origin of Species (2nd Edition)” should be equal, but “On the Origin of Species” and “Darwin's On the Origin of Species: A Modern Rendition” should not.
	* The author is the same, regardless of whether it is written as “Surname, Name” or “Name Surname”, e.g., “Charles Darwin” and “Darwin, Charles” are the same author.

 As you can see, there is no requirement for the isbn and year to be checked, i.e., two books may be “equal” even if their isbn and year of publication are completely different.

1. The library decides to include magazines in their catalogue. Create a new class named **Magazine** in package **uk.ac.sheffield.com1003.library.catalogue**. This class must extend class **CatalogueItem** with a numeric field named number (several numbers of a magazine can be published per year), which is taken as input argument to a constructor (i.e., public Magazine(int number)) and must override the toString() method to return a string with this format: "Magazine [title=..., year=..., number=..., copies=...]" (the dots should be replaced with concrete values and should not appear in the string). Extend **App.main** further with code that demonstrates that you can create a magazine object, add it to the library, and loan it.
2. After some time, the library realises that for diverse reasons, removing books from the catalogue becomes necessary (e.g., when books are damaged or stolen). Implement method **Library.removeItem** to allow them to delete a copy of an item given its *exact* title or ISBN and overload the method to include an additional integer argument **n** indicating how many copies should be deleted. When the last copy is deleted, the book instance should be destroyed, i.e., removed from the catalogue altogether. If the provided title does not exist, the method should throw a new instance of **ItemNotFoundException** (which you should have implemented for task 3).

You are expected to **document** your code using helpful comments and display **meaningful** messages to the console. The quality of the documentation and reporting of results will be assessed.

# Testing your classes

The test directory in your project includes some JUnit tests which are meant to help you assess that your solution is ‘correct’ and that you are working in the right direction. Your solution code will need to successfully pass the tests provided in these classes. However, note that the test suite provided does not fully test your solution, therefore just because your solution passes all these tests, it does not mean it is fully correct - you must make sure of this yourself. Feel free to add your own JUnit tests if you wish to (it will not earn you any additional marks), but do so in the test class named **MyTests.java** (not in **TestLibrary.java**). During marking, additional, more thorough tests (not included in the code provided) will be run on your code. Make sure to watch the supporting video showing how to verify that your code has passed all the JUnit tests provided.

# General Rules and Recommended Approach

* Start working on this problem sheet as soon as possible so that you can get help from the demonstrators and/or lecturers during the labs. Use the [‘Problem Sheet’ Discussion Board](https://vle.shef.ac.uk/webapps/discussionboard/do/forum?action=list_threads&course_id=_107949_1&nav=discussion_board_entry&conf_id=_346817_1&forum_id=_346107_1) to ask your questions, without sharing your solution.
* Study the code provided, and then complete the methods in the indicated classes (you need to create some classes). You have already been provided with the template methods and should not modify the [signatures](https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html) of these methods (i.e., you are not allowed to change any method’s name, return type or arguments).
* Read carefully the Javadoc comments provided in the template code, as they will specify the desired functionality that you are expected to implement.
* Watch the supporting video about how to run the tests in the provided test classes and confirm that your code passes all those tests. If not, please, check your code as you might have a bug there.
* You are encouraged to make good use of git by committing frequently and using descriptive commit messages. [Your commit history should give an idea of the work you have done in your solution](https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History). That said, you will not be marked down for not using git effectively.
* Once you are happy with your solution, you will need to do two things
	+ Make sure that you commit and push your code to your GitHub classroom repository before the deadline.
	+ [Go to Blackboard to submit your assignment](https://vle.shef.ac.uk/webapps/assignment/uploadAssignment?content_id=_7405546_1&course_id=_107949_1&group_id=&mode=cpview). **You do not need to upload any code to BlackBoard**, just add the link to your latest commit in the *Comments* section of your submission, e.g.,: **https://github.com/tuos-dcs-COM1003-23-24/problem-sheet-[YOUR\_GITHUB\_ID]/commit/xxxxxxxxxxxx**
* You must submit your solution before 17:00 on Monday, 18 Mar 2024.
* After the deadline has passed, standard late submission penalties and extenuating circumstances rules apply.

# Coding tips

Your code should be written using a good coding style, and you should note that *readability*is very important. Following the [Google Java guidelines](https://google.github.io/styleguide/javaguide.html) is recommended. In particular, you should:

· Use comments, but only when required (i.e. not excessively, but you should have a JavaDoc comment block at the head of each class).

· Use one code statement per line.

· Code your methods with as few lines as possible, and as many lines as needed. Ideally a complete method should be visible in a code editor without scrolling (see *Clean Cod*e for a longer discussion).

· Adhere to [naming conventions](https://google.github.io/styleguide/javaguide.html#s5-naming); class names in **UpperCamelCase**, method and variable names in **lowerCamelCase**, and constant names in **CONSTANT\_CASE**.

· Choose sensible, self-documenting, and descriptive names for all variables, methods, and classes.

· Use indentation consistently with either 2 or 4 whitespaces per indent (use whitespace because tabs can be interpreted differently on different operating systems).

· Before your code is checked, ask yourself whether the code is understandable by someone marking it.

# Assessment and hand-in procedure

This problem sheet is worth 20% of your mark for the Spring semester. Your marks will depend on developing a solution that:

* Compiles without needing modifications. This includes situations in which a character or characters in another language are used in the code[[1]](#footnote-0), or situations where an import to a non-existing package or class causes a compilation error. Solutions that do not compile, will automatically obtain a **zero mark**.
* Implements all the tasks listed in this handout.
* Passes all the tests provided in the test classes, which **must not** be modified.
* The code is appropriately documented[[2]](#footnote-1).
* The messages displayed in the console are meaningful, e.g., *“Book successfully added to catalogue”* and *“Could not find ISBN 9780099549482”* are meaningful messages, whereas *“I’m here”* or *“starting loop”* are not.
1. For those working on a computer with a different language charset, consider switching to an English charset to avoid potential problems, or confirm before submitting your code that your solution compiles and runs properly in a computer with an English charset (e.g. University computers). [↑](#footnote-ref-0)
2. As discussed in Week 5, it is difficult to be prescriptive as to what *appropriate documentation* is, as there is an element of subjectivity. A good question to ask yourself to check if you are on the right track is: ‘*If I gave my code to someone else, would they find it easy to understand and would they be able to maintain or extend it?*’ [↑](#footnote-ref-1)