Project T1 Week 7

- **Due** 16 Nov 2023 by 16:00
- **Points** 100
- **Submitting** a file upload
- **Available** 2 Oct 2023 at 0:00 - 20 Nov 2023 at 16:00

This assignment was locked 20 Nov 2023 at 16:00.

Introduction

This exercise forms the first part of the assessed coursework for the course. It is worth 50% of the total coursework mark (so 12.5% overall). Your submission must be entirely your own work. Do not submit code produced by or with another student or obtained from an AI program such as ChatGPT. If you submit anything other than your own work, you will receive a mark of zero and may be placed in the misconduct procedure. This is where you need to stop working cooperatively with other students if you've been doing that up to this point. It is expected that the work will be done in your own time, outside of lab sessions. But if you're fully up to date with the lab exercises you can use lab time to work on this assignment. Please do not ask demonstrators to answer questions about it. They cannot do so. Finally, bear in mind that interpretation of the specification is part of the exercise. If you feel it is ambiguous in some way, **do not ask for clarification.** Add code relevant to each potential interpretation, and include comments to explain what you have done.

How to Submit

Once you have completed the assignment you will need to submit it. This should be done electronically from within Canvas. You will need to archive (as a zip file) the BlueJ project folder containing your solution and submit it as a single file using the `Contributory E-submission' portal (via Project T1 Week 7). If you're a mac user, make sure to produce the zip file on a PC, and check that the zip file you generate will load back into BlueJ on a PC. Don't try to submit a zip file produced on a mac.

For this assignment you are required to write three Java classes: TextBook, Library and LibraryCard. Class TextBook represents information about a library text book. The Library and LibraryCard classes represent (university) libraries, and the cards that are issued to students to allow them to borrow text books:

- A text book has a title and contains a number of chapters that can be 'read' one chapter at a time.
- A new library has a book shelf that contains a collection of some number of text books.
- A library will issue a library card to anyone who wants to become a borrower at the library.

- A library card can be used to borrow text books from the library, up to a pre-specified number of times (after which, the card 'expires' and can no longer be used to borrow books).
- When a text book is borrowed, it is removed from the book shelf and issued to the borrower (so the library contains one less book).
- After being borrowed, a text book can be returned to the library, in which case it is put back on the book-shelf.

Full details of the classes are given below. Please note that it is important to follow the specification given for each class exactly (name of the class, signatures of the methods, etc.). The second assessed coursework assignment will build on the first, making various assumptions about the objects defined by the classes and the implemented behaviours.

The TextBook class.

Fields (Instance Variables):

The class should have three private fields: one of type String to represent the title of the text book, one of type int to represent the length of the book in chapters, and one of type int to represent the number of the last chapter to be read (i.e. the number of chapters read so far).

Constructor Method:

The class should provide a constructor method with two parameters, one of type String and the other of type int. The first parameter represents the title of the book and the second its length in chapters. The constructor should therefore initialise a TextBook object by setting the title of the book to the specified String, and the length of the book to the specified int. In addition, the number of the last chapter read should initially be set to zero (as nothing has been read at this point)

Other Methods:

A TextBook object should also have the following public methods associated with it:

- a method getTitle() that returns the String that represents the title of the book.
- a void method readNextChapter() that 'reads' the next chapter of the book (if this is possible): i.e. it should cause the 'last chapter' field to be increased by one, but only if there are chapters left to be read! If there are no more chapters left to read (i.e. the last chapter read was the final chapter of the book) then it should simply print out a suitable warning message.
- a boolean method isFinished() that returns true if there are no chapters of the book left to read and false otherwise.

- a void method closeBook() that resets the last chapter read field to zero (i.e. it is used to return the book to its initial state, ready for the next person to begin reading it).
- a void method describe() that prints out a simple description of the text book, with the book title and number of chapters left to read, e.g.: "Programming with Java with 4 chapters left to read"

The LibraryCard Class

Fields (Instance Variables):

Your implementation of the LibraryCard class should provide the following private fields:

- an int field that holds the limit on the number of books that the card can be used to borrow.
- an int field that holds the number of books that have been borrowed on the card so far.
- a String field that holds a card reference, e.g. "cardID 3", "cardID 27" or "cardID 94".

Constructor:

The constructor method should take two int parameters:

- the first is used to set the limit on the number of books that can be borrowed with the card.
- the second is used to set the card reference. Note: the card reference should be a String of the form "cardID NUM", where NUM is the actual value of the second formal parameter of the constructor.

The constructor should also initialise the 'number of books borrowed' field as 0 when the card is constructed.

Methods:

There are four methods in addition to the constructor:

a method with signature public void swipe(), which when invoked, increases the number of books borrowed on the card by one.   (This is not required to check whether the lending limit is breached.)

- a method with signature public boolean expired() which returns true if no more books can be borrowed on the card, and false otherwise.

- a method with signature public String getCardRef() that returns the value of the card reference field.

- a method with signature public void describe() that prints out a description of the card,   e.g. "Library card cardID 27 with 3 books left".

The Library Class

Fields (Instance Variables):

Your implementation of the Library class should declare the following private fields:

A field TextBook[] bookShelf which holds the library's collection of TextBook objects. (Note that the type of bookShelf is a fixed-length array of Textbook, not an ArrayList.)

- You will also find it useful to have an int field (called nextBook or something similar) which indicates the position on the bookShelf where the next TextBook object can be found (the next one available for borrowing when someone presents a valid library card).
- an int field representing the number of borrowers who have been issued with library cards.

Constructor:

The class should have a constructor with a single integer parameter which determines the maximum number of TextBook objects that may be held on the library's bookShelf.

Note: A problem here is to set up the Library object up so that it contains TextBook objects in an array. There are several steps in Java to setting up an array of objects:

1. Declare a variable of the appropriate type (in this case it is just the field bookShelf).
2. Create an array and store it in the variable (at this point it is an array of nulls).
3. Fill the array up with suitable objects (for which you will need to use a suitable loop statement)

A further issue is how to construct the TextBook objects that are put into the bookShelf array. One possibility is to give all the books the same title and number of chapters. A better possibility would be to give the books different titles, for example "text book 1", "text book 2","text book 3" and so one (the number of chapters can still be the same for each book). This is fairly simple to do as you can keep track of the number of each book as you go through the loop statement.

Methods:

The class has the following methods in addition to the constructor:

a method with signature public LibraryCard issueCard() that when invoked issues (i.e. returns) a new LibraryCard object. The method should increment the count of the number of library borrowers by 1. Note: to issue a new LibraryCard it is necessary to provide two int parameter values (see description of

LibraryCard constructor, above). The limit on borrowing can be fixed with a small int value, such as 5. However, the card identity number should be unique to each card, so you need to find a way of setting the actual parameter value to a different int value each time a new card is constructed. (Hint: think about numbering the cards sequentially, according to the number of borrowers. So, the first card issued might be given the reference "cardID 0", the next card "cardID 1", and so on.)

- A method with signature public TextBook borrowBook(LibraryCard card), which allows a book to be borrowed from the library if a valid (i.e. non-expired) LibraryCard is presented. If there are still books left to be borrowed in the library and the card object has not expired, then the method should return the next TextBook object from the bookShelf array (and otherwise, it should return the null object). The method should also 'remove' the TextBook object from the bookShelf (by setting the array location to null) and increment the field that indicates the location of the next book that may be borrowed. Finally, the method should ensure that the library card is 'swiped' to record that a book has been borrowed. You should also ensure that your method behaves sensibly if passed a null LibraryCard object.
- a method with signature public void returnBook(TextBook book) that allows a book to be handed back to the library. The method will need to put the book back in the bookShelf array. (Ideally, this method would also reduce the number of books borrowed on the relevant card. But for this coursework, you are not required to implement that feature.) It is important that the replacement of the book is done in a way that ensures there are no 'gaps' on the shelf. You may find that the pointer to the next book to be borrowed is again useful here. Once the TextBook has been put back in the array, the method will also need to change the 'next book' pointer appropriately.
- a method with signature public void describe() that, as usual, is responsible for printing out a description of the object. In this case, an example of the sort of message that should be printed out is: "The library has 15 books left on the shelf and has issued 7 library cards".

Testing

Having written the classes make sure that they work properly. Get one or more instances of the classes onto the BlueJ workbench and check that they behave the way that they should. Check what happens if you try to borrow a book with an invalid card, or when there are no books left on the book shelf. Make sure that you can only borrow as many books from the library as were put on the book shelf in the first place, and that a library card expires once it has been swiped up to its limit. Use BlueJ's inspect facility to check that text books borrowed from the library really are removed from the bookShelf array.

Layout and Code Style

It is important to make sure that you lay out your code properly. In particular make sure that everything is indented properly, and that opening and closing braces are lined up. Also make sure you use meaningful variable names and follow the Java naming conventions (i.e. follow capitalisation conventions). Note that no comments are required at this stage. We will be looking at commenting later.

Marking Scheme

To get any marks at all, the student must submit a zip file of the folder containing the BlueJ package specified. Marks are awarded according to the code, even if it fails to compile. Provided the submission is made correctly, the following scheme applies.

50 marks awarded in all.

class TextBook: (12 marks overall)

Fields: 2 marks for correct work.

Constructor: 2 marks for correct work.

Methods: 8 marks in all for correct work. Methods getTitle() and closeBook() are worth 1 mark each. Method isFinished(), describe() and readNextChapter() carry 2 marks each. All marks lost for a method if it is completely incorrect, incoherent or missing. Lose 1 mark if isFinished() does the right thing but redundantly. Similarly, lose 1 mark for describe() if it uses more than a single print statement. For readNextChapter(), the conditional expression should not allow a chapter to be read if the book is finished.

class LibraryCard: (worth 15 marks overall)

Fields: 2 marks for correct work.

Constructor: 3 marks for correct work.

Methods: 10 marks for work as per the spec. Methods describe() and getCardRef() are worth 2 marks each; methods swipe() and expired() are worth 3 marks each; Marks lost for use of incorrect return types, or over-use of conditional statements in method expired().

class Library: (worth 23 marks overall)

Fields: 2 marks for correct work. Marks lost for missing out a field or failing to make it private or of the correct type.

Constructor: 4 marks for correct work. Marks lost for failing to initialise fields appropriately. Note that a loop is needed to initialise the bookShelf[] array. Marks lost for getting the bounds on the loop wrong or failing to build a new TextBook object to assign to each location. One mark lost for failing to give each book a unique title (same number of chapters is ok).

Methods: 17 marks for correct work. Methods returnBook() and describe() earn 3 marks each; method issueCard() is worth 4 marks, and method borrowBook(LibraryCard card) is worth 7. Marks lost for a method if it is completely incorrect or missing. Marks also lost for failing to implement a method as described in the brief (check that borrowers field is incremented when a card is issued, and that the LibraryCard gets a unique card reference number). Marks lost if the returnBook(TextBook book) method doesn't put books back in the way specified in the brief.

Layout & Commenting:

Code should be laid out clearly and systematically. Indentation should be used, and a consistent and sensible bracketing convention adopted throughout. Marks lost for weird, unsystematic, inconsistent or obscure layout. Comments are not expected.

This assignment is submitted through Canvas

Please note:

- It is your responsibility to ensure you submit the correct work for your assignment: please check carefully that any files are your final work before you submit them.
- In making a submission, you are declaring that your work contains no examples of academic misconduct, such as plagiarism, collusion or fabrication of results.
- You can submit and resubmit work up to your deadline.
- After your deadline has passed:
    - If you have already submitted work your last submission before the deadline will be marked and any further submission you make will not be considered.
    - If you have not yet submitted work you will have a further **24 hours** to make a submission. This late submission will normally incur a penalty. Only the first work submitted during this period will be marked; any further work you submit during this period will not be considered.

Submission

Submitted!

8 Nov 2023 at 5:46

Grade: 100 (100 pts possible)

Graded anonymously: no

Comments:

Textbook Class - 12 out of 12 LibraryCard class - 15 out of 15 Library Class - 23 out of 23

xxxxx, 21 Nov 2023 at 21:24

Project T1 Week 11

- **Due** 15 Dec 2023 by 16:00
- **Points** 100
- **Submitting** a file upload
- **Available** 2 Oct 2023 at 0:00 - 22 Dec 2023 at 16:00

This assignment was locked 22 Dec 2023 at 16:00.

This exercise forms the second part of the assessed coursework for *Introduction to Programming*. It will be worth 50% of the total coursework mark, and thus 12.5% of the marks overall. Your submission must be entirely your own work. Do not submit code produced by or with another student, or obtained from an AI program such as ChatGPT. If you submit anything other than your own work, you will receive a mark of zero and may be placed in the misconduct procedure. If you've been working cooperatively with another student in the lab sessions, this is where you need to stop doing that. It is expected that the work will be done in your own time, outside of lab sessions. But if you're fully up to date with the lab exercises you can use lab time to work on the assignment. Please do not ask demonstrators to answer questions about it. They cannot do so. Finally, bear in mind that interpretation of the specification is part of the exercise. If you feel it is ambiguous in some way, **do not ask for clarification.** Add code relevant to each potential interpretation, and include comments to explain what you have done.

How to Submit

Once you have completed the assignment you will need to submit it as a zip file. As for the first assignment, this will be done from within Canvas. You will need to zip the BlueJ project folder containing your solution and any associated files (e.g., documentation) and submit it using the `Project T1 Week 12' link in the Assignments section. To do this, right-click on the name of the folder containing the project and do `Send to' choosing `Compressed (zipped) Folder'. Then submit the .zip file that gets created. Make sure the zip file you submit contains the folder containing the BlueJ package and the API documentation, and nothing else.

If you've been using a **mac** computer to do the coursework, you should transfer the project to your file area on the lab PCs before zipping it. Don't use the `Compress' option on the mac. This does create a zip file but it also inserts package instructions and additional file structure which causes problems for the marking process.

The Assignment

To start this project you will need a BlueJ project containing the TextBook class, the Library class, the LibraryCard class from the first coursework. You can either use your submission for the first assignment or you can make use of the sample answer provided (see Week 8 section). There is no penalty for making use of the sample answer as part of the second assignment. Marks for the second assignment will be based entirely on the additional classes specified below.

For this assignment you are required to add two further Java classes to the project, one called Student and one called College. In addition, you will be expected to:

- make your project 'stand-alone' (i.e. it should be possible to compile and run your project without the use of BlueJ);
- document your code appropriately using javadoc conventions. You're only required to document the two new classes, but if you want to add comments for the three original classes, you're welcome to do that. You will get no extra marks for this however.
- provide evidence of unit testing using the framework integrated into BlueJ.

The New Classes: Student and College

Full details of the behaviour required from Student objects and College objects, and the fields and methods needed to achieve this are given below.

Instances of Student represent college students. Students can join their college Library to obtain a LibraryCard, which they can then use to borrow TextBooks for their studies. The Students read the TextBooks that they borrow and return them to the Library when they are finished. A Student is finished studying when their LibraryCard expires and they can't get hold of any more books.

An instance of the College class simulates a college, which has a Library and an arbitrary number of Students. Time advances in discrete steps. At each step a random Student wakes up, studies for a bit (for example, by borrowing a TextBook from the Library, or reading a TextBook if they already have one) and then goes back to sleep again. Students may eventually finish their studies, in which case they are removed from the college.

class Student

Instance Variables (Fields)

Each Student object should have the following instance variables:

- a field of class String, which stores the name of the student
- a field of class Library, which stores the Library object representing the college library which the student joins
- a field of class LibraryCard to store the library card issued to the student on joining the Library.
- a field of class TextBook to store the TextBook object representing the current text book borrowed by the Student from the Library. It has the value null if they do not have a TextBook.

Constructor Method

The Student class should provide a constructor method with the signature public Student(String name, Library library), where the parameter name is a string representing the student's name, and library is a Library object, representing the library that a new student joins.

The constructor initialises the fields as follows:

- the first parameter is used to set the value of the String field representing the Student's name
- the second parameter is used to set the value of the Library field, representing the library that the Student joins.
- the field of class LibraryCard is initialised as the LibraryCard issued to the Student by the Library.
- the TextBook is initially assigned the value null.

Other Methods

The Student class should provide the following additional methods:

1. a method with signature public boolean finishedStudies() which should return true if the student does not have a TextBook to read, but can't borrow one because the LibraryCard has expired. Otherwise, it should return false.
2. a method with signature public void study(), which causes the student to study, and has the following behaviour:

    - if the student does not currently have a TextBook then the student tries to borrow one from the library;
    - if the student does have a book and it is not finished, then the student reads one chapter of the book;
    - otherwise, the student has a book that is finished and so closes it and returns it to the library. (Note that after this the student should not have a book.)

3. A method with signature public void describe(), which prints out messages about the student, e.g. about whether or not they have a TextBook and if they do have book, whether or not the book is finished or they are still reading it. For example, invoking the method may give messages such as:
   "Student Bill does not have a book and library card cardNum4 has an allowance of 4 books."
   "Student Mary has a book text_0 with 4 chapters left.
   The student is reading the book."
   "Student Bill has a book text_2 with 0 chapters left.
   The book is finished"
   And so on. You will need to think how to make use of the describe() methods of the other classes (TextBook, LibraryCard) in order to generate sensible messages about the student and what is going on.

class College

Instance Variables (Fields)

Each College object should have the following instance variables:
a field of type ArrayList<Student>
a field of type Library
a field of type Random

Constructor Method

The College class should provide a constructor method with two int parameters representing the number of Students enrolled at the college and the number of TextBooks in the library, respectively.

The constructor should initialise the fields as follows:

- the Library field should be initialised to a new Library object that contains the specified number of TextBooks.
- the ArrayList<Student> field should be set to hold a new ArrayList holding the specified number of Student objects. Note: When automatically creating Student objects you will need to name them in some way. A good way of doing this is to give them names such as Student1, Student2, Student3 etc since you can then generate the names automatically while looping (as done when intialising the bookShelf array in the Library class*).*

- the Random field should be set to hold an object of class Random that can be used to enable random students to be selected by the class code.

Other Methods

The College class should define the following additional methods:

1. a method with signature public void describe() which prints out a simple description of the state of the College, such as

   "The college currently has 9 hard-working students"
   "The library has 8 books on the shelf and has issued 7 cards"

2. a method with signature private void nextStep() for internal use by this class only (hence the method is private) which carries out the following actions:
   - if there are no Students left at the College, then the method should just print out a message such as "Everything has gone quiet."
   - otherwise, a Student should be selected at random from those still at the College. If the selected Student has finished their studies, then they should be removed from the College and a suitable message printed out (e.g. "The student has graduated and left the college."); otherwise, the Student should study.
3. a method with signature public void runCollege(int nSteps), which repeatedly carries out the following actions (exactly nStep times):
   1. print out the number of the step being carried out (e.g. "Step 12")
   2. describe the state of the College (i.e. invoke the describe() method); and
   3. invoke the nextStep() method.

Making Your Program Stand-alone

To complete the coding of your project you should add a main method, with signature public static void main(String[] args), to the College class that will allow the project to be run without BlueJ. As a minimum, this method should first construct a College object with some fixed number of Students and TextBooks. It should then invoke the runCollege(...) method for some specified number of steps. (Note that you will need to choose sensible numbers for the various parameters).

For maximum marks, the main method should be constructed to allow the various parameter values (i.e. number of Students, number TextBooks, number of steps to run) to be specified as command line parameters using the String[] args parameter of the main method. For example, you might invoke the project with the command:

```
java College 10 15 100
```

meaning, set up a College with 10 Students and 15 TextBooks in the Library and then run for 100 steps.

Layout and javadoc Documentation

Make sure you lay out your code properly. In particular make sure that everything is indented properly, and that opening and closing braces are lined up. Also make sure you use meaningful variable names and follow the Java naming conventions (e.g. the capitalization convention).

For this assignment you are expected to document your code using javadoc conventions. You should comment your code appropriately, making use of javadoc tags where possible. To get the marks you only need to document the newly written classes Student and College.

Finally, you will need to generate and submit the resulting API documentation along with the rest of your project. (Documentation generated by BlueJ is stored in the same project folder, so this will be automatically included when you zip the project folder.)

Unit Testing

You should test your code as you go along to make sure that everything works properly. In BlueJ, this is most easily done by creating instances of your classes, and testing their methods one by one. For this coursework you are also required to make use of the JUnit tools built in to BlueJ. You should construct a test class for both College and Student, and use these to create (i.e., record) methods that fully test the functionality of the methods you have written .

Marking Scheme

class Student (13 marks overall)

Fields: 2 marks for correct work.

Constructor: 3 marks for correct work:

Methods: 8 marks overall for correct work. Method finishedStudies() is worth just 2 marks, while study() and describe() are each worth 3 marks. Marks are lost for a method if it is completely incorrect or missing. Marks also lost for incorrect logic in study() method (the order in which the conditions are executed may be significant). Marks lost if the describe() method fails to make use of the describe() methods of the TextBook and LibraryCard classes.

class College (12 marks overall)

Fields: 2 marks for correct work.

Constructor: 3 marks for correct work.

Methods: 7 marks overall for correct work, not including main(). Methods runCollege(int nSteps) and describe() are worth 2 marks each, while nextStep() carries 3.

main method (10 marks overall)

Worth 10 marks overall. Minimally, the main method should set up a new College (with appropriate values for number of students and number of text books) and call the runCollege() method for some appropriate number of steps. A minimal solution with these values hard-wired is worth a maximum of 2 marks. More sophisticated solutions will allow at least some of the parameters (e.g. number of students in the lab, number of time-ticks) to be set on the command line. Up to 3 marks can be earned in this way. Advanced solutions may provide some command line processing to allow for variable numbers of parameters, use of default values, etc. Up to 5 marks can be earned in this way.

Layout & Comments (5 marks overall)

Up to 5 marks for sensible commenting and layout. Indentation should be used and a consistent and sensible bracketing convention adopted throughout. The code should be commented throughout using javadoc conventions. The class should have a brief comment with @author and @version tags; each method should have an associated comment above its signature with appropriate use of @param and @return, as necessary. Comments should not be overly verbose, but they should say enough to understand what the various classes and methods are for. The code should have associated documentation produced using javadoc. Marks are lost for weird, unsystematic, obscure, or verbose things or failing to put any comments in at all.

BlueJ Test Classes (10 marks overall)

Up to 10 marks for providing test classes and methods for Student and College. Evidence of sensible test cases for all of the methods is required. For any method with a return value, there should also be a test method that validates this.

This gives a possible maximum total of 50 marks.

This assignment is submitted through Canvas
Please note:

- It is your responsibility to ensure you submit the correct work for your assignment: please check carefully that any files are your final work before you submit them.
- In making a submission, you are declaring that your work contains no examples of academic misconduct, such as plagiarism, collusion or fabrication of results.
- You can submit and resubmit work up to your deadline.
- After your deadline has passed:
    - If you have already submitted work your last submission before the deadline will be marked and any further submission you make will not be considered.
    - If you have not yet submitted work you will have a further **7 days** to make a submission. This late submission will normally incur a penalty. Only the first work submitted during this period will be marked; any further work you submit during this period will not be considered.

Submission

Submitted!

30 Nov 2023 at 11:52

Grade: 71 (100 pts possible)

Graded anonymously: no

Comments:

** If your submission was late, your final grade will be automatically reduced in the system. This means that your final grade will be different from the grade mentioned in the overview. ** Layout & Comments (5 marks) : 4.5 out of 5 - 0.5 mark lost for comment styles that are inconsistencies. Class Student (13 marks) - 6 out of 13 -Constructor : 1 mark lost for not properly writing code. It seems you are still confused about how and where to use the keyword 'this.' - finishedStudies() : 1 mark lost for not properly checking if the student's library card has expired or if there are no more books to read. - study() : 3 marks lost for an invalid - describe() : 2 marks lost for not following the specification from the requirement. Class College (22 marks) - 17 out of 22 - Construtor : 1 mark lost for incorrectly initialising a new the ArrayList<Student> - nextStep() : 1 mark lost for not following the specification from the requirement. - main() : 3 marks lost for not following the specification from the requirement and the method does not implement correctly. BlueJ test (10 marks) - 8 out of 10 - 2 marks lost for not fully following the specification from the requirement. Overall: 35.5 out of 50 (71%) Excellence! You've made a good progress, but there are still a few areas that need improvement. Despite that, your work is commendable. Keep up the good work.

Xxxxx Oh, 27 Dec 2023 at 23:41