**Lab Assignment 3:**

**Library Catalog System**

|  |
| --- |
| **Due Date**  **(a two-week LA)** |
| **2/24/16 11:59pm – For Wednesday Lab Sections**  **2/25/16 11:59pm – For Thursday Lab Sections** |

**Objectives**

* To practice designing proper inheritance relationships
* To practice overriding superclass methods in subclasses

**Problem Specification**

You are to write a Java application which can be used to keep track of a library’s collection of materials. The collection will be read in from an input file, and users will be able to display a list of materials and check them out by call number.

**Library Materials**

* Books (checked out for 21 days)
  + Book Title
  + Author
  + Genre
* Periodicals (checked out for 7 days)
  + Periodical Title
  + Volume
  + Issue
  + Subject

All library materials have a **call number**, and **can be checked out**. The **date checked out** should be the current date, and the **due date** depends on the type of material. When displaying the item, show the specific information, along with the call number, whether or not the item is checked out, and if so, the checked out and due dates (see example output below).

**Input File**

* The first line is the number of books in the library’s collection.
* The second line is the number of periodicals in the library’s collection.
* Each line after that contains a single library item (fields separated by commas)
  + Books (indicated by the letter B)
    - Fields: *Call number, Book title, Author, Genre*
  + Periodicals (indicated by the letter P)
    - Fields: *Call number, Periodical title, Volume, Issue, Subject*

**Example Input:**

2

2

B,C124.S17,The Cat in the Hat,Dr. Seuss,Children’s Literature

P,QJ072.C23.37.4,Computational Linguistics,37,4,Computational Linguistics

P,QJ015.C42.55.2,Communications of the ACM,55,2,Computer Science

B,F380.M1,A Game of Thrones,George R. R. Martin,Fantasy Literature

The application should exhibit the following functionality (see the sample output below):

* Read the contents of the library’s collection from a file when the application starts.
  + The file name should be hardcoded
* Allow the user to choose from a menu of options:
  + Display collection
    - Displays the full list of materials
  + Check out materials
    - Ask the user for the call number, and then find the matching item.
    - If an item is already checked out, do not allow the user to check it out and display a message indicating it is not available.
    - Set the checked out date and due date accordingly.
    - Display the item information
  + Quit
    - Exits the application
* Allow the user to continue making requests until s/he selects the Quit option.

**Example Output:**

------------- Menu -------------

1. Display collection
2. Check out materials
3. Quit

------------------------------------

Please choose an option: 1

Book Title: The Cat in the Hat

Author: Dr. Seuss

Genre: Children's Literature

Call Number: C124.S17

Checked Out: NO

Periodical Title: Computational Linguistics

Volume: 37

Issue: 4

Subject: Computational Linguistics

Call Number: QJ072.C23.37.4

Checked Out: NO

Periodical Title: Communications of the ACM

Volume: 55

Issue: 2

Subject: Computer Science

Call Number: QJ015.C42.55.2

Checked Out: NO

Book Title: A Game of Thrones

Author: George R. R. Martin

Genre: Fantasy Literature

Call Number: F380.M1

Checked Out: NO

------------- Menu -------------

1. Display collection
2. Check out materials
3. Quit

----------------------------------

Please choose an option: 2

Enter the call number: F380.M1

Book Title: A Game of Thrones

Author: George R. R. Martin

Genre: Fantasy Literature

Call Number: F380.M1

Checked Out: YES

Date Out: 02/08/16

Date Due: 02/29/16

------------- Menu -------------

1. Display collection
2. Check out materials
3. Quit

----------------------------------

Please choose an option: 2

Enter the call number: F380.M1

Item is not available.

------------- Menu -------------

1) Display collection

2) Check out materials

3) Quit

--------------------------------

Please choose an option: 3

Your application **MUST** make use of a proper inheritance relationship. It should have a **superclass** representing a general library item, and **subclasses** representing the specific types of materials included in the collection (books and periodicals). Any common data members or methods between the subclasses should be defined in the superclass. Specific functionality, such as additional data members or overriding methods, should be defined in the specific subclass.

Your application should store objects of a similar type in an array of the appropriate type. You should therefore have two arrays to store the items: one for book objects and another one for periodicals.

**Hints**

1. Working with dates in Java is very easy to do! You need to use the **GregorianCalendar** class (the name just refers to the type of calendar familiar to most of the western world). You can get the current date by instantiating a new **GregorianCalendar** object. You can copy the date using the clone method, and modify the date by using the add method. The add method below adds 50 days to the date.

import java.util.Calendar;

import java.util.GregorianCalendar;

...

dateCheckedOut = new GregorianCalendar();

dateDue = (GregorianCalendar)dateCheckedOut.clone();

dateDue.add(Calendar.DAY\_OF\_YEAR, 50);

String.format("Date Out: %tD\n", dateCheckedOut);

The **%tD** format specifier will print the date in *mm/dd/yy* format

Output for this code snippet will be (assuming today’s date is 02/08/16):

Date Out: 03/29/16

1. Remember that the split method can be used to split a string using a specified delimiter, and returns an array of strings. The following line of code splits a string delimited by commas.

String[ ] line = inputString.split(",");

**Design** **Requirements**

***You are required to finish the design report before you leave the lab.***

*Note: Lab reports**will be worth* ***40%*** *of the total LA3 grade.*

Your lab report should contain the following two parts.

1. **Basic Structure**

This section of your report should be done using UML diagrams. Your diagrams should show the relationships between classes (sub- and superclasses, classes implementing interfaces, association between classes (if any).

Your project should have a class **LibraryItem** which implements the **ILibrary** interface. This interface is provided and no changes should be made to it. All methods in the interface must be implemented by class LibraryItem.

LibraryITem has two subclasses named **Book** and **Periodical** respectively.

**NOTE**: Any common data members or methods between the subclasses should be defined in the superclass. Specific functionality, such as additional data members or overriding methods, should be defined in the specific subclass.

There is a **Controller** class which implements the **IController** interface. This class is used by the main class to run the program.

The code for the main class (which has only the main method) is provided and should be used as provided, without modifications. This serves as a test class for your program.

The interfaces and the main class are provided below.

**package** edu.wmich.cs1120.LA3.LibraryCatalog;

**import** java.io.IOException;

**public** **interface** IController {

/\*\*

\* Displays the collection of library items to the screen

\*/

**public** **void** displayCollection();

/\*\*

\* Requests for the call number from the user, uses the findItem()

\* method to check if that item exists in the library, and if it does

\* calls the checkOut() method for that item and prints out the item

\* that has been checked out.

\*/

**public** **void** checkoutMaterials();

/\*\*

\* Searches in both the array of books and the array of periodicals

\* for the book with the call number received as a parameter.

\* **@param** callNum The call number of the item requested by the user

\* **@return** The requested item, or 'null' if item does not exist.

\*/

**public** ILibrary findItem(String callNum);

/\*\*

\* Displays the menu options to the user.

\*/

**public** **void** showMenu();

/\*\*

\* Reads data from the input file and stores the items in the

\* appropriate array.

\* **@param** fileName The name of the input file.

\* **@throws** IOException Included in case input file is not found.

\*/

**public** **void** readInput(String fileName) **throws** IOException;

} // End of interface IController

**package** edu.wmich.cs1120.LA3.LibraryCatalog;

**import** java.util.GregorianCalendar;

**public** **interface** ILibrary {

/\*\*

\* Sets the boolean value checkedOut to true, and

\* initializes the dateChecked out attribute (a

\* GregorianCalendar object).

\*/

**public** **void** checkOut();

/\*\*

\* Generates a string with the details of the library item

\* whose call number has been input by the user (see example output)

\* and returns that string.

\* If the user wants to check out the library item, the string to be

\* returned also includes information that the item has been checked out,

\* the date it was checked out, and the due date for the item.

\* **@return**

\*/

**public** String toString();

/\*\*

\* Returns the call number of this object.

\* **@return** the callNumber

\*/

**public** String getCallNumber();

/\*\*

\* Returns true or false depending on if this item has been checked out.

\* **@return** the boolean value for isCheckedOut

\*/

**public** **boolean** isCheckedOut();

/\*\*

\* Returns the date this item was checked out.

\* **@return** the dateCheckedOut

\*/

**public** GregorianCalendar getDateCheckedOut();

/\*\*

\* Returns the date this item is due to be returned.

\* **@return** the dateDue

\*/

**public** GregorianCalendar getDateDue();

/\*\*

\* Sets the dateDue to the parameter received.

\* **@param** dateDue the dateDue to set

\*/

**public** **void** setDateDue(GregorianCalendar dateDue);

} // End of ILibrary interface

**package** edu.wmich.cs1120.LA3.LibraryCatalog;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** LA3Assign {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

Scanner keyboard = **new** Scanner(System.***in***);

IController control = **new** Controller(keyboard);

control.readInput("input.txt");

String response = "";

**boolean** quitFlag = **false**;

**while** (!quitFlag) {

control.showMenu();

response = keyboard.nextLine();

System.***out***.println();

**switch** (response) {

**case** "1":

control.displayCollection();

**break**;

**case** "2":

control.checkoutMaterials();

**break**;

**case** "3":

quitFlag = **true**;

**break**;

}

}

keyboard.close();

}

} // End of main class

1. **Pseudocode**

Write pseudocode for all the required methods.

**Additional Requirements**

**Coding Standards**

You must adhere to all conventions in the CS 1120 Java coding standard. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming projects, packages, classes, variables, method parameters and methods.

**Javadoc**

You must include **Javadoc** comments and generate a Javadoc document for your project. Points will be deducted in the following cases:

* Javadoc comments are included but the Javadoc document is not generated.
* No Javadoc comments are included at all in the project.

**Testing**

Test your application with other input data besides the example given here, to make sure it works for all data. You should also test it for different sequences of user input (displaying the collection, checking out various materials, displaying the collection again, trying to check out items that are already checked out, etc.).

**Assignment Submission**

* Generate a .zip file that contains all your files, including:
  + Program Files
  + Any input or output files
* Submit the .zip file to the appropriate folder in elearning.

**NOTE**: The elearning folder will be inaccessible after the due date/time.