Managing a Library Card System
CS 5004 Object Oriented Design

Due to Canvas by end of day, May 31.

1. **Goals:**
    - Practice designing a system
    - Create more complex methods
    - Use design methodologies such as composition, abstraction, and inheritance
    - Work more with different data types

2. **In Recitation:**

    Composition and Inheritance are two major design techniques that help cut down on repetitive code and ensure that a program makes use of all that object-oriented design can empower us to do. During recitation, we will guide you through the initial set-up of a parent class with a child class and an object instance that leverages composition with another object.

3. **Instructions:**

    Checking out books from a library is exciting! But all libraries need to be able to track who has checked out which book, when it is due back, where it ought to go, and what fines or late fees might be associated with a given borrower. To that end, we will help our fictional librarians design a library card system.

    Part 1: The Library Card System:

Given classes/methods:

1. Class Person - This class represents a person who has a first name, last name, and a date of birth. There are getter methods to get one's first name, last name, and date of birth. There is also a *toString()* method that allows a Person's first and last name to be displayed.
2. Class LibraryCard - This class is used to represent a library card a person can hold and use. You must first make a constructor for this class. It will take in no parameters, but it will be able to keep track of how many total library cards there are (numCards), and what number card a person's library card is. Adding an attribute related to total fees might be beneficial. You must have the following classes (think about the appropriate data types to use with each):

    getCardNumber(), which returns the card number of that particular library card.

    getTotalFees(), this will return the total fees associated with a library card, if late fees for each BorrowedBook are not null.

borrowBook(Book book), checks if the book is available and puts it into one of three available borrowedBook spots.

returnBook(BorrowedBook book), the late fees must be updated first (if applicable), and then it can be returned.

makePayment(float amount), this subtracts from the totalFees and returns that value.

3. Class Book - This class represents a book. A book can have a title, author, and a standard value associated with only that book. Like the Person class, there will be getter methods and a *toString()* method that will have to be filled in based on the directions provided within the starter code.

Classes and methods to create:

1. Class BorrowedBook - This class will inherit from the book class. First off, import java.time.LocalDate. You will have to utilize outside resources to figure out what this is for. You should use the LocalDate object as an issueDate and a dueDate in the class parameters. Do not forget, a borrowed book can also accrue late fees. The constructor for this class should have Book book and, LocalDate issueDate. It must also inherit the attributes from the parent class. When trying to determine the books dueDate, use the method *.plusDays(long daysToAdd*) that comes with the import you put in at the start. You must have the following methods:

   LocalDate getIssueDate(), this returns the issueDate.

   LocalDate getDueDate(), this returns the dueDate.

   getLateFees(), (what data type would be best to use for this one?) this method must check if late fees should be applied, if so, update the attribute and return the updated value.

4. Driver - This is where you will run your code. You will have to demonstrate the code works as expected by using manual testing methods. (Your driver is typically the “main” class that runs your code).

   <u>Part 2: Testing and Documentation</u>

   For each method:

   - Design the signature of the method.
   - Write Javadoc-style comments for that method.
   - Write the body for the method.
   - Write your JUnit tests for your methods. Make sure you document the tests with comments.

4. **Reflection:**

   Each assignment must include a short reflection. It should take you no more than 15 minutes to write. Your reflection should be two paragraphs that speak to the following questions, though you do not need to answer each of them directly:
   - What gave you the hardest problem in creating the program associated with this lab assignment?
   - What is something that you feel you understand better in Java now? Is there anything you encountered that you were able to solve a problem to, but you don't understand how you did it? Are there questions you have about object-oriented design after completing this assignment?

Submission:

Submit your program as a single zip file named: `YourName_Lab3.zip`

Submission checklist:

- ☐ Did you include adequate comments?
- ☐ Did you include comments at the top of each file?
- ☐ Did you name your files appropriately?
- ☐ Does your code compile?
- ☐ Did you remove any package lines generated by your IDE?
- ☐ Did you take care of any warnings presented by your IDE?