Object-Oriented Programming Lab#9, Spring 2022

Today's Topics

- Inheritance
- encapsulation
- method override
- method overload
- subclass polymorphism
- abstract class
- Add project reference

A Library Management System

Develop a **Library management system** where user can **checkout/borrow** items (books / magazines/ published papers) for a specific time for study, and **return** the item to the library within the specified time. Also the application will be used to **search** for specific items, view the list of items matching specific criteria and many more functionalities as described below. The application will be used by both the **librarian** and **member** of the library. A library keeps different types of items which has some common characteristics and some item-specific characteristics. For simplicity, we will only consider only the following 3 types of items in our system.

1. **Book** – Book item has a title, category, one or multiple authors, published date and publisher's name.
2. Published **Paper** or Publication – Published paper could be a journal paper or conference paper (similar to the category of Book). Like Book, published paper also has a title, one or multiple authors, published date, and the name of the conference or journal
3. **Movie** – Each movie has a title (name), category, one or multiple directors (similar to authors of Book and Published Paper), and published date.

The system will have the following functionalities.

1. There are 2 types of users for this system; **a librarian and member**. A user can log in as a librarian or member. For simplicity, we can **skip** the log-in part and add an option or button for logging in as a **librarian** or **member**. A member will have less functionality than the librarian.
2. The system will have the following functionalities. Most of the functionalities will be

available to both types of user.

a. Add more items to the library (Librarian only)
b. Add new member information to the system(Librarian only)
c. Librarian can **check-out** an item on behalf of a member or the member can check-out an item himself,
d. Extend the check-out time. This feature is allowed only once per checkout.
e. Check-in a checked-out item for a member. **(**Librarian only**)**
f. Search for items using title and author
g. Search for items with a specific published year.
h. View the list of items of a specific category
i. View all items (optional: organized by a specific category)
j. View the **Checkout record** of an **item (Librarian only)**
k. Librarians can view the **Checkout record** of a **member** whereas a member can only view his checkout record.

*What you need to do: (**Note: Do not use default package**)*

You need 2 eclipse/intelliJ IDEA projects for this Lab.

Create a Project name Library (and do the following)

1. Create a class name **CheckOutRecord**:
   a. Add 5 private instance variables; ***memberId***, ***itemId***, ***checkOutTime***, ***expectedCheckInTime***, *and* ***checkInTime***
      - *All 3 time related parameters are* **java.time.LocalDateTime** type**.**
   b. Implement a parameterized constructor with just ***memberId*** and ***itemId***. Inside the constructor, set the ***checkOutTime*** to LocalDateTime.Now() and ***expectedCheckInTime*** to ***checkOutTime.plusWeeks(1)***

   Add the following method

   c. ***public void returnItem()*** – set the ***checkInTime*** to now.
   d. Add getter/setter as needed.

e. Override the toString**()** method and return the string in the format *"**itemId \tmemberId\tcheckOutTime** \t **checkInTime"** here the String value of checkout and checkin time should be printed.*

Code to convert time to string [Use this in toString() method]

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
String appTime = checkOutTime.format(formatter);
```

Note: `DateTimeFormatter` is under `java.time.format` package.

2. Create an **Member** class:
   a. Add 3 **private** instance variables; ***memberId***, ***name***, *and* ***chekOutRecords*** *(an ArrayList of type* ***CheckOutRecord*** *to store the records of check-out/check-in).*
   b. Implement parameterized constructor.

   Add the following methods inside the class:

   c. public void addCheckOutRecord(CheckOutRecord record)

   - Inside the method, add a record to ***chekOutRecords***.

   d. Add getter/setter as needed.

3. Create an **abstract** class **Item**:
   a. Add 6 private instance variables; ***itemId***, ***title***, ***category***, ***authors (an ArrayList of type String)***, ***publishDate***, ***isCheckedOut*** *and* ***chekOutRecords*** *(an* ***ArrayList*** *of type* ***CheckOutRecord*** *to store the records of check-out/check-in).*
   b. Implement parameterized constructor and pass parameters for itemId, title, category, author and publish date.

   Add the following methods inside the class:

   c. public void checkOut(String memberId)

   - Inside the method, do nothing if the ***isCheckedOut*** is true. If it is false do the following

- set the ***isCheckedOut*** to true

- Create an object of **CheckOutRecord** class with ***memberId*** set to the parameter, ***itemId*** set to attribute ***itemId***. Add the object to the ***chekOutRecords*** list.

d. public void checkIn()

- Inside the method, do nothing if the ***isCheckedOut*** is false. If it is true do the following

  - set the ***isCheckedOut*** to false

  - Retrieve the last record of the ***chekOutRecords*** list. Call the ***returnItem***() method.

e. public void extendCheckOut()

- Inside the method, do nothing if the ***isCheckedOut*** is false. If it is true do the following

  - Retrieve the last record of the ***chekOutRecords*** list. If the **expectedCheckInTime** is within one week of ***checkOutTime***, set the ***expectedCheckInTime*** one week later using the **setExpectedCheckinTime(LocalDateTime)** method**.** Print "Already extended once. Cannot extend again" message otherwise.

f. public boolean isAnAuthor(String author)

- This method returns true if the ***author*** parameter is available in the ***authors*** attribute. You do not need to do an exact match, rather use the "***contains***" method of ***String*** class for comparison.

-

g. Add **public getter** method for all attributes and setter method for ***isCheckedOut*** attribute.

h. Override toString() method

- From the method, return a String in the format "itemId-title-{comma separated authors}-publishDate-isCheckedOut".

i. **public ArrayList< CheckOutRecord > getCheckOutRecords()**

- This method will return the **checkOutRecords** variable.

j. **public CheckOutRecord getLatestCheckOutRecords()**

- This method will return last item of the **checkOutRecords** variable

4. Create a **Book** class:
    a. Make this class a **subclass** of **Item** class.
    b. Add **an additional private** instance variable ***publisherName***.
    c. Implement parameterized constructor.
    d. Override ***the toString()***
    - Call the ***toString()*** method of the parent class and then concatenate "-***publisherName***".

    e. Add getter/setter method for the additional attributes.

5. Create a **Publication** class:
    a. Make this class a **subclass** of **Item** class.
    b. Add **an additional private** instance variable ***publisherName***.
    c. Implement parameterized constructor.
    d. Override ***the toString()***
    - Call the ***toString()*** method of the parent class and then concatenate "-***publisherName***".

    e. Add getter/setter method for the additional attributes.

6. Create a **Movie** class:
    a. Make this class a **subclass** of **Item** class.
    b. Implement parameterized constructor.

7. Now create a class name "**Library**" which will mimic a real Library that holds a list of items **(book, paper, movie and many more items)**. Use an Array or **ArrayList** to hold the list of Items. The class will have two attribute ***ArrayList<Item> items*** and **ArrayList<Member> members**. Add the following methods to the class.

a. private void addItem(Item item)

- Inside the method, add the ***item*** object to the ***items*** list.

b. public void addBook(*String itemId*, String *title*, String *category*, ArrayList<String>*authors,* DateTime *publishDate, String publisherName****)***

- Inside the method, create a ***Book*** object using the parameter provided and add the account to the list using ***addItem (Item)*** method.

c. public void addPublication(*String itemId*, String *title*, String *category*, ArrayList<String>*authors,* DateTime *publishDate, String publisherName****)***

- Inside the method, create a ***Publication*** object using the parameter provided and add the account to the list using ***addItem (Item)*** method.

d. public void addMovie(*String itemId*, String *title*, String *category*, ArrayList<String>*authors,* DateTime *publishDate****)***

- Inside the method, create a ***Movie*** object using the parameter provided and add the account to the list using ***addItem (Item)*** method.

e. ***private Item findItem(String itemId)***

- This method will loop through the list of the **Items** (***items***) and find the item that has matching ***itemId*** as the parameter. If the matching **Item** is available return the object otherwise return null.

f. ***public ArrayList<Item> findItems(String itemTitle, String author)***

This method will loop through the list of the **Items** (***items***) and find the items with title containing the ***itemTitle*** parameter and has ***author*** parameter as one of the authors (use isAnAuthor(String) method) of the item. If the matching **Items** are found return those as an ArrayList, return null otherwise.

g. ***public ArrayList<Item> findItems(String itemType)***

This method will loop through the list of the **Items** (***items***) and find the items that are of ***itemType(Book, Publication, Movie)*** type. If the matching **Items** are found return those as an ArrayList, return null otherwise.

h. public void checkOut(String itemId, String memberId)

- Inside the method call ***findItem(String)*** to find the item with matching ***itemId***. Also call ***findMember(String)*** to find the member with matching ***memberId.*** If both the item and the member is found do the following
    - Call ***checkOut(memberId)*** method using the item object.
    - Call the ***getLatestRecord***() using the item object.

- Add that record to the member's ***checkOutRecords*** list using **addCheckOutRecord** () method.

i. public void extendCheckOut(String itemId)

- Inside the method call ***findItem(String)*** to find the item with matching ***itemId*** and then call ***extendCheckOut()*** method of that object.

j. public void checkIn(String itemId)

- Inside the method call ***findItem(String)*** to find the item with matching ***itemId*** and then call ***checkIn()*** method of that object.

k. public ArrayList<CheckOutRecord> getCheckOutRecords(String itemId)

- Inside the method call ***findItem(String)*** to find the item with matching ***itemId*** and then call ***getCheckOutRecords()*** method of that object and return.

Add the following member specific methods.

a. public void addMember(String id, String name)

- Inside the method, create a ***Member*** object using the parameter provided and add the member to the ***members*** list.

b. ***private Member findMember(String memberId)***

- This method will loop through the list of the **Members** (***members***) and find the member that has matching ***memberId*** as the parameter. If the matching **Member** is available, return the object otherwise return null.

Now create a new project LibraryApp (and do the following).

1. Add project reference of the previous project.
2. Create an **application class** (that has the main method) named "**LibraryApp**" which will have the **main** method. Create an object of the Library class.
   - In the main method, ask if the user is a **librarian** or a **member**.
   - If the user is a librarian, display the following menu to the user. Depending on the option, take required input/data from the user and call the appropriate method.
     - Input '1' to add a new Item.

       You need to provide a submenu to create different types of Items. So, you have to ask what **type of item** he wants to add. Depending on the user response, take further inputs and call the appropriate method to add the item.

        - Input ‘2’ to search for an item with itemId.
        - Input ‘3’ to search for an item with title and author.
        - Input ‘4’ to checkout an item.
        - Input ‘5’ to extend the checkout time.
        - Input ‘6’ to check-in an item.
        - Input ‘7’ to view the checkout record of an item.
        - Input ‘8’ to view the checkout record of a member.
        - Input ‘9’ to display the list of specific types of items.
        - Input ‘10’ to display the list of all items.
        - Input ‘11’ to add members.
        - Input ‘0’ to exit the system.
    - If the user is a member, ask him/her to enter the memberId. Display the following menu to the user and take necessary action.
        - Input ‘2’ to search for an item with itemId.
        - Input ‘3’ to search for an item with title and author.
        - Input ‘4’ to checkout an item.
        - Input ‘5’ to extend the checkout time.
        - Input ‘7’ to view the checkout status of an item.
        - Input ‘8’ to view his/her checkout records.
        - Input ‘9’ to display the list of specific types of items.
        - Input ‘10’ to display the list of all items.
        - Input ‘0’ to exit the system.