MTU

Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

OBJECT ORIENTED PRINCIPLES

ASSIGNMENT 3: A JAVA APPLICATION FOR MANAGING A STORE.

Declaration of Authorship

I, ___YOUR NAME___, declare that the work presented in this assignment titled ‘A Java Application for Managing a Store’ is my own. I confirm that:

- This work was done wholly by me as part of my BSc. (Hons) in Software Development, my Msc at Munster Technological University.
- Where I have consulted the published work and source code of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this assignment source code and report is entirely my own work.

On ___DATE___

Signature:

Java Application Description.

This Java application provides an interactive, text menu-based, session for managing a public library. The functionality of the application includes:

- Adding/removing users and items to the library (in the case of the items, both books and music albums are supported);
- Displaying the info of a user/item;
- For users to borrow and return items of the library;
- For a manager to impose/remove fines to users (based on late return of items);
- For the library content to be loaded/stored to a text-based database.

Technical Difficulty: OOP Concepts Demonstrated in the Java Application.

1. **Primitive and Reference Variables.**
   - The class Item.java has the field releaseYear (an int and, therefore, primitive variable) and the field currentBorrowed (a Borrow and, therefore, reference variable).
2. **Classes and Objects.**
   - The class User.java models a user of the library, and the method LibraryImp.java::addUser creates a new User object newUser in line 62.
3. **Encapsulation.**
   - The class User.java has a private field isFined, and public methods getIsFined and setIsFined methods to access/update the field from other classes.
4. **Aggregation.**
   - The class Item.java has a private field currentBorrowed, an object of type Borrow.
5. **Inheritance.**
   - The classes User.java and Item.java inherit from Agent.java.
6. **Class Hierarchy.**
   - The classes Book.java and MusicAlbum.java inherit from Item.java. Therefore, there is a class hierarchy, where User.java and Item.java are siblings, and MusicAlbum.java is a grandchildren of Agent.java.
7. **Static Polymorphism (overloading).**
   - The class MyMain.java has two versions of the method selectIntOption, each of them with a different signature.
8. **Dynamic Polymorphism (overwriting).**
   - The classes User.java and Item.java overwrite the method toString, specified in the class Object any Java class automatically inherits from.
9. **Abstract Class.**
   - The class Item.java is declared abstract, as it contains an abstract method numDaysToBorrow. The method must, therefore, be overwritten by any class inheriting from Item.java (as is the case in the classes Book.java and MusicAlbum.java).

10. Interface.

- The class Library.java is an interface, modelling the management of a public library (via methods as addUser, removeUser, etc). The interface is implemented in the class LibraryImp.java.

11. User and Developer Isolation.

- Abstract Datatypes isolate the *what* (what represents this data and what operations can we do with it) from the *how* (how is this data internally represented and how is each operation internally implemented).
- See the UML diagram on this appendix:
    - Let's assume the class MyMain.java was implemented by Programmer1. She can look at Library.java and then create a variable of type Library to use all its functionality (the methods addUser, removeUser, etc.), without knowing how all this functionality is internally implemented.
      All she needs is to use the object of type Library for her own application, programmed in the methods of the class MyMain.java. In this case, her application is an interactive text menu for using a library.
    - Let's assume the class LibraryImp.java was implemented by Programmer2. She knows how to represent internally a library (via a number of fields) and how to implement each of the methods offered. On doing so, she also implements the rest of classes (Agent.java, User.java, etc.)
      On programming the library implementation and the rest of classes, she makes sure other programmer can create a variable of type Library to use all its functionality. But Programmer2 does not know the type of application programmer1 is creating (maybe an interactive text menu, a graphic-based app, a web-based one, etc).

12. Upcasting.

- The method LibraryImp::loadItemsFromDisk creates the variable Item myItem in line 675. The variable is there assigned to a newly created Book object (line 681) or to a newly created MusicAlbum object (line 694).

13. Static Fields and Methods.

- The class LibraryImp.java has a static field nextId. Therefore, the field does not belong to a single object of the class, but to all objects of the class.
  Indeed, if I were to program the Java application again, I would put this field in the class Agent.java, where it definitely fits better.
- The class User.java has a method isUserInUsersList, to compute whether any of the users of a list contains a concrete id. As a public static method, it can be called from any class without the need of a concrete User object, just by using the prefix User.isUserInUsersList.

14. Final Fields, Methods and Classes.

- The class User.java has a final field name, as once it is defined, it cannot be modified.
- The class Agent.java has a final method getId, so that no other class inheriting from Agent.java (for example, User.java or Item.java) can overwrite the method and compute the id in a different way.

15. Data Structures.

- The class LibraryImp.java has a field usersList, representing the list of users of the library. The list is represented as an ArrayList of User objects.

16. Java Generics.

- The class LibraryImp.java has a field usersList as an ArrayList<User> and a field itemsList as an ArrayList<Item>. The use of Java Generics allows to have lists of different types.

17. Downcasting.

- As Item.java is an abstract class, and LibraryImp.java has a field itemsList as an ArrayList<Item>, when we access the concrete objects of the list for doing something with them, we implicitly refer to the child class methods. For example, when we use Item myItem and then myItem.numDaysToBorrow in line 392 of LibraryImp.java.

18. Exception Handling.

- The method MyMain::selectIntOption handles the exception of a user inputting by keyboard a value that is not an integer (as expected by sc.nextInt in line 110). The instruction is placed in a try block; if something goes wrong, for example if the user enters instea a String value, then the block catch is executed, instead of making the whole application crash.

19. File Reading and Writing.

- The method LibraryImp::loadContentFromDisk reads the content from a text file and loads it into the Java Application (specifically, into the ArrayList<User> and ArrayList<Item> lists of the library manager).
- The method LibraryImp::saveContenToDisk writes the content from a list of users or items to a text file (specifically, the ArrayList<User> and ArrayList<Item> lists of the library manager).

20. Default Constructor and Copy Constructor.

- At the time of writing this appendix, I realise I have not included a copy constructor or default constructor in the Java application, but this is something that could have been added also :(

UML Design: Java Application.

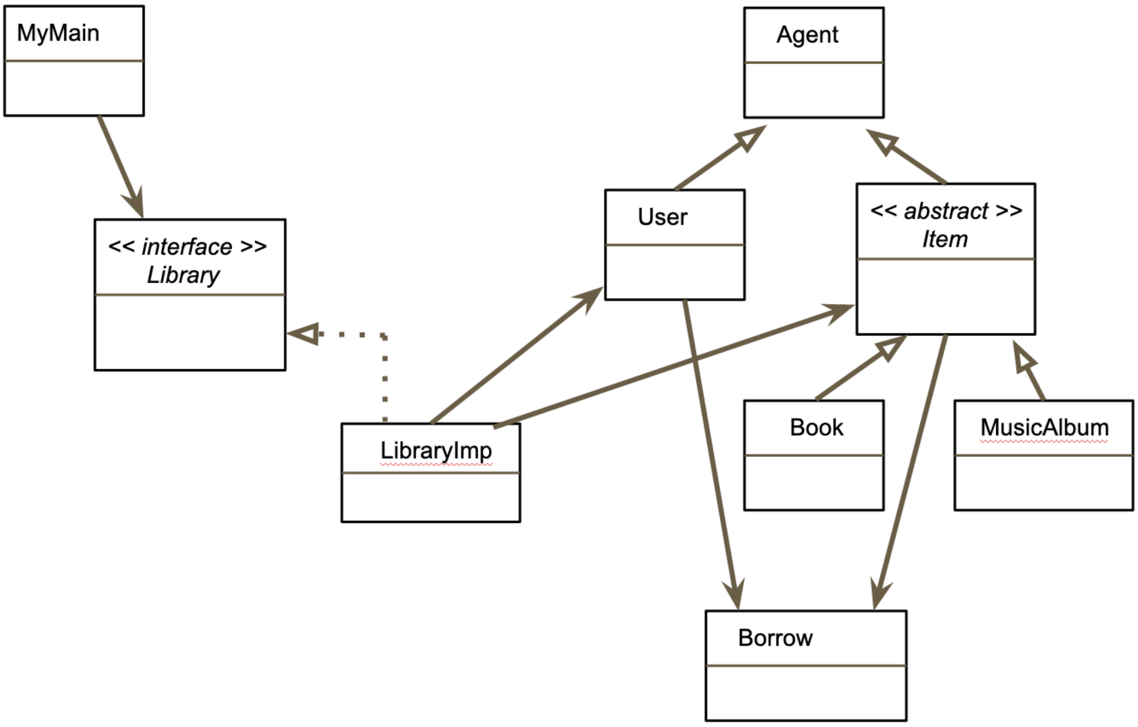
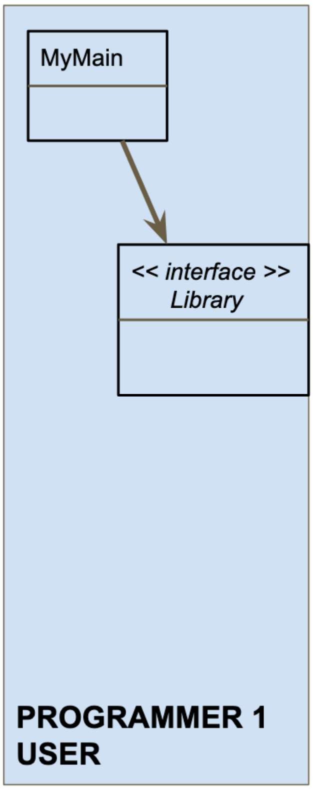
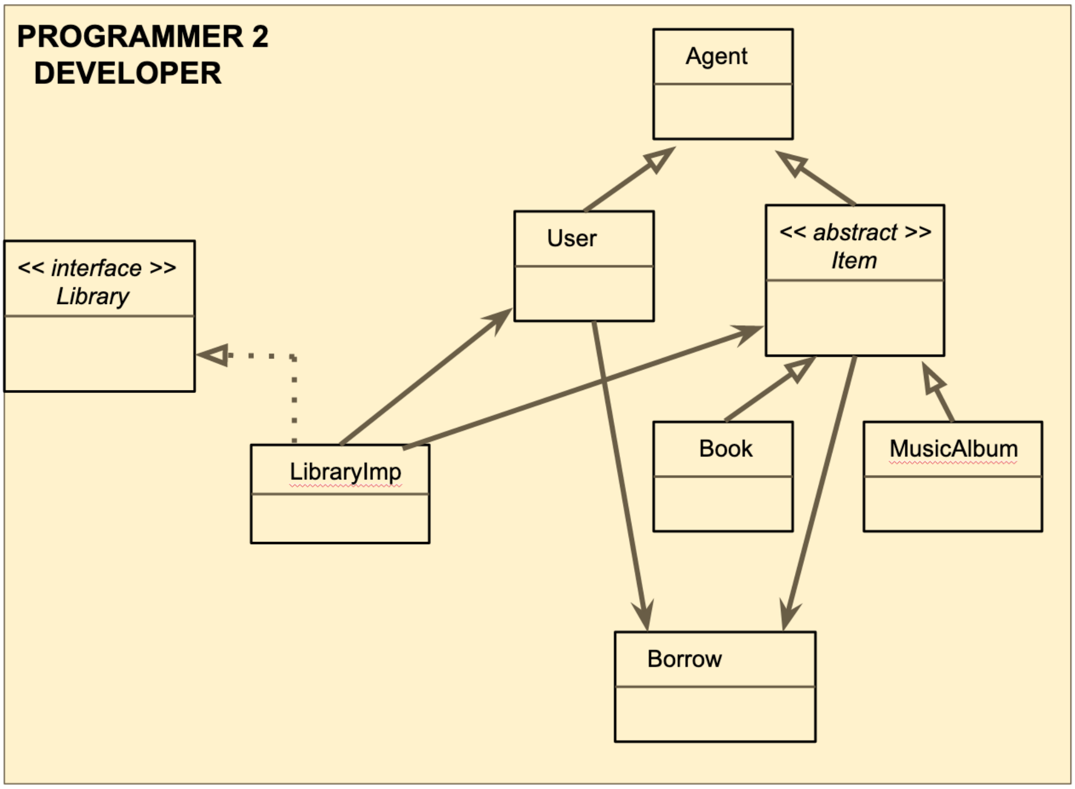


Diagram Legend

MyClass	Java Class
⟶	Uses
· · · · ▷	Implements
—▷	Inherits from


MyMain
<< interface >>
Library
PROGRAMMER 1
USER


PROGRAMMER 2
DEVELOPER
Agent
User
<< abstract >>
Item
<< interface >>
Library
LibraryImp
Book
MusicAlbum
Borrow

Testing the Java Application.

The functionality of the application is tested in MyMain.java via an interactive, text menu-based, session. On it, we can select among a range of different commands to test the different functionality of the library.

Of course, an alternative way of testing the application would be the one used in most of the code example this semester. That is, to create a number of test methods in MyMain.java, and make the main method to use an option integer variable and a switch clause to select which test method to try on each run of the Java application.