Interfaces

Overview

In this lab, you will add a "reservation service" to the library system that you worked on in the previous lab. The reservation service will allow members to reserve items that are currently borrowed by another member.

You will define a `Reservable` interface to identify the types of items that can be reserved (it's common for libraries to allow types of item to be reservable, but to disallow some other types of item from being reserved).

Source folders

Student project: `StudentInterfaces`
Solution project: `SolutionInterfaces`

Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Defining an interface
2. Implementing the interface
3. Writing client code
4. Additional suggestions

Exercise 1: Defining an interface

Open your student project and take a quick look at the code we've provided as the starting point for this lab. This is the sample solution from the "Inheritance" lab.

Now define a new interface named `Reservable`, which represents "reservable capability". The `Reservable` interface should have the following methods:

- `isReserved()`
  Takes no parameters, and returns a `boolean` to indicate whether this item is currently reserved.
- `canBeReservedFor()`
  Takes a `Member` object as a parameter, and returns a `boolean` to indicate whether the member is allowed to reserve this item.
- `reserveItemFor()`
  Takes a `Member` object as a parameter, and returns a `boolean` to indicate whether the reservation succeeded.

Exercise 2: Implementing an interface

In the library system, not all books will be reservable. Popular books (such as those by bestselling authors) will be reservable because they are likely to be in high demand, but many books don't warrant this administrative overhead (e.g. would you bother reserving a book on Welsh Football Greats? I would, but most people probably wouldn't …. ☺).

With this in mind, define a new class named `ReservableBook`, to represent books which are reservable. `ReservableBook` must extend `Book` and implement the `Reservable` interface.

The `ReservableBook` class should have the following additional members, to extend the capabilities of the `Book` class:

- An instance variable indicating the member who has reserved the book (only one reservation is allowed at a time for books).
- Suitable constructor(s).
- An implementation of `isReserved()`, which returns a `boolean` to indicate whether there is a reserver already for this item.
- An implementation of `canBeReservedFor()`, which allows a reservation if:
    - The member passes the basic rules for borrowing a book (as specified in the `canBeBorrowedBy()` method).
- An implementation of `reserveItemFor()`, as follows:
    - If the book is currently borrowed by someone, is not currently reserved, and can be reserved by this member, reserve it and return `true`.
    - Otherwise, just return `false`.
- An override for the `returnItem()` method:
    - First, invoke superclass behaviour for this method, to perform the "normal" business rules for a returned item.
    - Then add some new code to test if the book is currently reserved by a member. If it is, immediately instigate a "borrow" operation for that member. In other words, as soon as a reserved book is returned, the reserver automatically becomes the new borrower. (Don't forget to set the "reserver" instance variable to `null` afterwards).

Exercise 3: Writing client code

Open `MainProgram.java`, and uncomment the existing code. Then add some new code to test the reservation service.

Suggestions and requirements:

- Create a `Reservable[]` array, and populate it with all the *reservable* items (i.e. loop through the items and use `instanceof` to see if an item implements the `Reservable` interface. If it does, store a reference to the item in the `Reservable[]` array).
- Test whether you can reserve a `ReservableBook` that isn't yet borrowed. This should not be allowed.
- Test whether you can reserve a `ReservableBook` that has already been borrowed. This should succeed. Furthermore, when the book is eventually returned, it should be automatically borrowed by the member who reserved it.

Exercise 4 (If time permits): Additional suggestions

Implement reservation capabilities in the `DVD` class (all DVDs are reservable, because they're all very popular). A DVD can be reserved by up to 5 members at a time; use an array here.

Suggestions and requirements:

- Implement `isReserved()` to return a `boolean` to indicate whether there are any reservers for this item at the moment.
- Implement `canBeReservedFor()` to allow a reservation if:
    - The member passes the basic rules for borrowing a DVD (as specified in the `canBeBorrowedBy()` method).
- Implement `reserveItemFor()` as follows:
    - If the DVD is currently borrowed by someone, there are fewer than 5 reservations already, and the DVD can be reserved by this member, reserve it and return `true`. Be careful to ensure that reservations are stored in the order they are placed.
    - Otherwise, just return `false`.
- Override `returnItem()` as follows:
    - First, invoke superclass behaviour for this method.
    - Then add some new code to test if the DVD is currently reserved. If it is, immediately instigate a "borrow" operation for that first reserver. (Don't forget to remove the reserver from the list of reservers).
- Extend the `main()` code to test the reservation capabilities for DVDs.