Inheritance

Overview

In this lab, you will implement a simple library system. You will define an inheritance hierarchy to represent different types of items that can be borrowed from a library (books, DVDs, etc.).

Source folders

Student project: `StudentInheritance`
Solution project: `SolutionInheritance`

Roadmap

There are 5 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. Getting started with the library system
2. Defining a base class
3. Defining subclasses
4. Writing the client application
5. Additional suggestions

Exercise 1: Getting started with the library system

In the student project, take a look at the pre-written `Member` class, which represents a member in a library. The `Member` class has the following members:

- Instance variables containing the member's name and age, plus an integer indicating how many items the member has currently borrowed.
- A constructor, which initializes the instance variables.
- A `toString()` method, which returns a textual representation of the member's details.
- Methods named `borrowedItem()` and `returnedItem()`, which increment and decrement the "items borrowed" count respectively (you'll call these methods whenever an item is borrowed or returned by the member).

Exercise 2: Defining a base class

Define an `Item` class, which will be the base class for all the different types of item in the library. `Item` should be an `abstract` class - why?

The `Item` class should have the following instance variables:

- Title (a `String`, initialized in a constructor).
- Date borrowed (a `Date`, `null` initially to indicate it's not borrowed yet).
- Current borrower (a reference to a `Member` object, `null` initially).

The `Item` class should have the following instance methods:

- `isBorrowed()`
  Returns a `boolean` to indicate whether the item is currently borrowed (test the "current borrower" field to see if it's `null`).
- `canBeBorrowedBy()`
  Takes a `Member` object as a parameter, and returns a `boolean` to indicate whether this member is allowed to borrow this item. Returns `true` by default (subclasses might override this method, with different rules for whether a member can borrow specific types of item)
- `borrowItemBy()`
  Takes a `Member` object as a parameter, and returns a `boolean` to indicate whether the borrow succeeded. The implementation of this method should follow these guidelines:
  - The fundamental rule is that an item can't be borrowed if someone has already borrowed it!
  - There are additional constraints on who can borrow what, as defined by the `canBeBorrowedBy()` method, so invoke this method. If the method returns `true`, set the item's instance variables to record the borrower and the date borrowed. Also, invoke `borrowedItem()` on the `Member` object, to increment the number of items borrowed by this member.
  - Return `true` or `false`, to indicate whether the borrowing succeeded.
- `returnItem()`
  Takes no parameters, and returns `void`. The implementation of this method should follow these guidelines:
  - Call `returnedItem()` on the item's borrower, to decrement the number of items borrowed by that member.
  - Set the item's instance variables to record the fact that the item is no longer borrowed.
- `toString()`
  Returns a textual representation of the item's details, indicating whether the item is currently on loan (and to whom).

Exercise 3: Defining subclasses

Define `Book` and `DVD` classes, which inherit from the `Item` base class.

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

- Instance variables for the book's author, ISBN, and genre. Implement the genre as an enum (allowable options are `Children`, `Fiction`, `NonFiction`).
- Suitable constructor(s).
- An override for the `canBeBorrowedBy()` method. The policy for books is that any member can borrow fiction and non-fiction books, but only children (age <= 16) can borrow children's books.
- An override for `toString()`, to return a textual representation of a book (including all the basic information in the `Item` base class, of course).

The `DVD` class should have the following additional members, to extend the capabilities of the `Item` base class:

- Instance variables for the DVD playing time (in minutes) and classification. Implement the classification as an enum (allowable options are `Universal`, `Youth`, `Adult`).
- Suitable constructor(s).
- An override for the `canBeBorrowedBy()` method. "Universal" DVDs can be borrowed by anyone; "youth" DVDs can be borrowed by anyone 12 or over; and "adult" DVDs can be borrowed by anyone 18 or over.
- An override for `toString()`, to return a textual representation of a DVD (including all the basic information in the `Item` base class).

Exercise 4: Writing the client application

In `MainProgram.java`, write a `main()` method to test the classes in your hierarchy.

Suggestions and requirements:

- First, create some `Member` objects with various names and ages.
- Then declare an `Item[]` array variable, which is capable of holding any "kind of item".
- Create some `Book` and `DVD` objects and place them in the array.
- Borrow some books and DVDs. Test the rules that govern whether a member is allowed to borrow a book.
- What happens if a member attempts to borrow an item that is already borrowed by someone else? Clearly this shouldn't be allowed, but does your application enforce this rule? Where would you write the code to make this test…?
- Write a loop to iterate through all the items and display each one. Verify that the correct `toString()` method is called on each item, thanks to polymorphism.

Exercise 5 (If time permits): Additional suggestions

In the `Item` class, define an `abstract` method named `dateDueBack()`. The method return type should be `Date`. The purpose of the method is to indicate when the item is due to be returned. This policy is different for each type of item, hence the reason for declaring it `abstract`.

In the `Book` class, override `dateDueBack()` so that it returns a date 21 days after the date the book was borrowed (you can use the `Calendar` class to do date arithmetic ☺).

In the `DVD` class, override `dateDueBack()` so that it returns a date 7 days after the date the DVD was borrowed.

Add some code in `main()` to test `dateDueBack()` policy.