Programming Paradigms 2014-15

Object-Oriented Programming with Ruby

Purpose of this assignment

- To exercise your knowledge of classes and objects in Ruby

General idea of the assignment

You are going to implement software for your local library. The user of the software will be the librarian, who uses your program to issue library cards, check books out to members, check books back in, send out overdue notices, and open and close the library.

Library rules:

- You must have a library card in order to check out books.
- You can have at most 3 books checked out at a time.
- Books are due one week (7 days) after being checked out

Details

In a number of methods, we will make a distinction between what the program *appears* to do (its *behaviour*), and what it *actually* does (it's *implementation*). For example, when we "*issue a library card*" to a member, we don't create some kind of a "`library_card object`" and give it to a member object; the implementation will achieve the same result by simply recording the member's name in a dictionary. The visible behaviour is the same: A person who has been "*issued a library card*" will be able to check out books. Don't let yourself be confused by the terminology.

You should not need any additional classes, but if you would like additional methods, feel free to write (and test!) them.

Classes and methods

Create the following classes.
All classes should be defined in a file named `library.rb`, and all test classes should be in a separate file named `library_test.rb`.

`class Calendar`

We need to deal with the passage of time, so we need to keep track of dates. Ruby has a perfectly good `Time` class, but to keep things simple, we will just use an integer to keep track of how many days have passed. This class needs only a couple of methods:

`initialize()`

The constructor. Sets the date to 0. You should create **one and only one** Calendar object; time is the same for everyone.

`get_date()`

Returns (as an integer) the current date.

`advance()`

Increment the date (move ahead to the next day), and returns the new date.

`class Book`

A book has these attributes (instance variables): `id`, `title`, `author` (only one author per book), and `due_date`. The due date is `nil` if the book is not checked out.

`initialize(id, title, author)`

The constructor. Saves the provided information. When created, the book is not checked out.

`get_id()`

Returns this book's unique identification number.

`get_title()`

Returns this book's title.

`get_author()`

Returns this book's author.

`get_due_date()`

Returns the date (as an integer) that this book is due.

`check_out(due_date)`

Sets the due date of this Book. Doesn't return anything.

`check_in()`

Sets the due date of this Book to `nil`. Doesn't return anything.

`to_s()`

Returns a string of the form `"id: title, by author"`.

`class Member`

A member is a "*customer*" of the library. A member must have a library card in order to check out books. A member with a card may have no more than three books checked out at any time.

`initialize(name, library)`

Constructs a member with the given *name*, and no books. The member must also have a reference to the Library object that he/she uses.

`get_name()`

Returns this member's name.

`check_out(book)`

Adds this `Book` object to the set of books checked out by this member.

`give_back(book)`

Removes this `Book` object from the set of books checked out by this member. (Since members are usually said to "*return*" books, this method should be called `return` !)

`get_books()`

Returns the set of **`Book`** objects checked out to this member (may be the empty set.

`send_overdue_notice(notice)`

Tells this member that he/she has overdue books. (What the method *actually* does is just print out this member's name along with the notice.)

`class Library`

This is the "master" class. It has a number of methods which are called by the "librarian" (the user), not by members. The librarian does all the work. Since these methods (other than the constructor) will be typed in by the librarian, all the parameters must be strings or numbers, not objects. Furthermore, to reassure the librarian that his/her action has worked, all these methods should return a result, for example, "Library card issued to Lenny Bruce."

`initialize()`

Constructs the Library object (you should create exactly one of these).

- Read in, from a file named **`collection.txt`**, a list of **`(`*`title`*`,`*`author`*`)`** tuples, Create a `Book` from each tuple, and save these books in some appropriate data structure of your choice. Give each book a unique id number (starting from 1, not 0). You may have many copies of the "same" book (same title and author), but each will have its own id.
- Create a `Calendar` object (you should create exactly one of these).
- Define an empty dictionary of members. The keys will be the names of members and the values will be the corresponding Member objects.
- Set a flag variable to indicate that the library is not open.
- Sets the current member (the one being served) to **`nil`**

`open()`

If the library is already open, raises an **`Exception`** with the message **`"The library is already open!".`** Otherwise, starts the day

by advancing the `Calendar`, and setting the flag to indicate that the library is open. (). Returns: The string **"Today is day *n*."**

find_all_overdue_books()

Prints a nicely formatted, multiline string, listing the names of members who have overdue books, and for each such member, the books that are overdue. Or, the string **"No books are overdue."**.

issue_card(name_of_member)

Issues a library card to the person with this name. However, no member should be permitted to have more than one library card.

Returns either "Library card issued to *name_of_member*." or "*name_of_member* already has a library card.".

Possible Exception: "The library is not open.".

serve(name_of_member)

Specifies which member is about to be served (and quits serving the previous member, if any). The purpose of this method is so that you don't have to type in the person's name again and again for every book that is to be checked in or checked out. What the method should actually do is to look up the member's name in the dictionary, and save the returned Member object in a data variable of this library.

Returns either **"Now serving *name_of_member*."** or **"*name_of_member* does not have a library card."**.

Possible **Exception**: **"The library is not open."**

find_overdue_books()

Prints a multiline string, each line containing one book (as returned by the book's **to_s** method), of the books that have been checked out by the member currently being served, and which are overdue. If the member has no overdue books, the string "`None`" is printed.

May throw an **Exception** with an appropriate message:

- **"The library is not open."**
- **"No member is currently being served."**

check_in(*book_numbers) # * = 1..n of book numbers

The book is being returned by the current member (there must be one!), so return it to the collection and remove it from the set of books currently checked out to the member. The **book_numbers** are taken from the list printed by the `search` command. Checking in a `Book` will involve *both* telling the Book that it is checked in *and* returning the Book to this library's collection of available Books.

If successful, returns "*name_of_member* **has returned** *n* **books."**.

May throw an **Exception** with an appropriate message:

- **"The library is not open."**
- **"No member is currently being served."**
- **"The member does not have book** *id***."**

search(string)

Finds those **Book**s whose title or author (or both) contains this string. For example, the string **"tact"** might return, among other things, the book *Contact*, by Carl Sagan. The search should be case-insensitive; that is, **"saga"** would also return this book.

Only books which are currently available (not checked out) will be found. If there is more than one copy of a book (with the same title and same author), only one will be found. In addition, to keep from returning too many books, require that the search string be at least 4 characters long.

Returns one of:

- **"No books found."**
- **"Search string must contain at least four characters."**

- A multiline string, each line containing one book (as returned by the book's `to_s` method.)

`check_out(*book_ids) # 1..n book_ids`

Checks out the book to the member currently being served (there must be one!), or tells why the operation is not permitted. The `book_ids` could have been found by a recent call to the `search` method. Checking out a book will involve *both* telling the book that it is checked out *and* removing the book from this library's collection of available books.

If successful, returns `"n books have been checked out to name_of_member."`.

May throw an `Exception` with an appropriate message:

- `"The library is not open."`
- `"No member is currently being served."`
- `"The library does not have book id."`

`renew(*book_ids) # 1..n book_ids`

Renews the books for the member currently being served (by setting their due dates to today's date plus 7) or tells why the operation is not permitted.

If successful, returns `"n books have been renewed for name_of_member."`.

May throw an `Exception` with an appropriate message:

- `"The library is not open."`
- `"No member is currently being served."`
- `"The member does not have book id."`

`close()`

Shut down operations and go home for the night. None of the other operations (except `quit`) can be used when the library is closed.

If successful, returns the string `"Good night."`.

May throw an `Exception` with the message `"The library is not open."`

`quit()`

The mayor, citing a budget crisis, has stopped all funding for the library. Can happen at any time. Returns the string `"The library is now closed for renovations."`.

Checking books in and out is slightly complex, so here is what the librarian needs to know:

To check books in:

1. If you have not already done so, `serve` the member. This will print a numbered list of books checked out to that member.
2. `check_in` the books by the numbers given above.

To check books out:

1. If you have not already done so, `serve` the member. You can ignore the list of books that this will print out.
2. `search` for a book wanted by the member (unless you already know its `id`).
3. `check_out` zero or more books by the numbers returned from the `search` command.
4. If more books are desired, you can do another `search`.

Submission

As part of your portfolio; see the website for due dates.