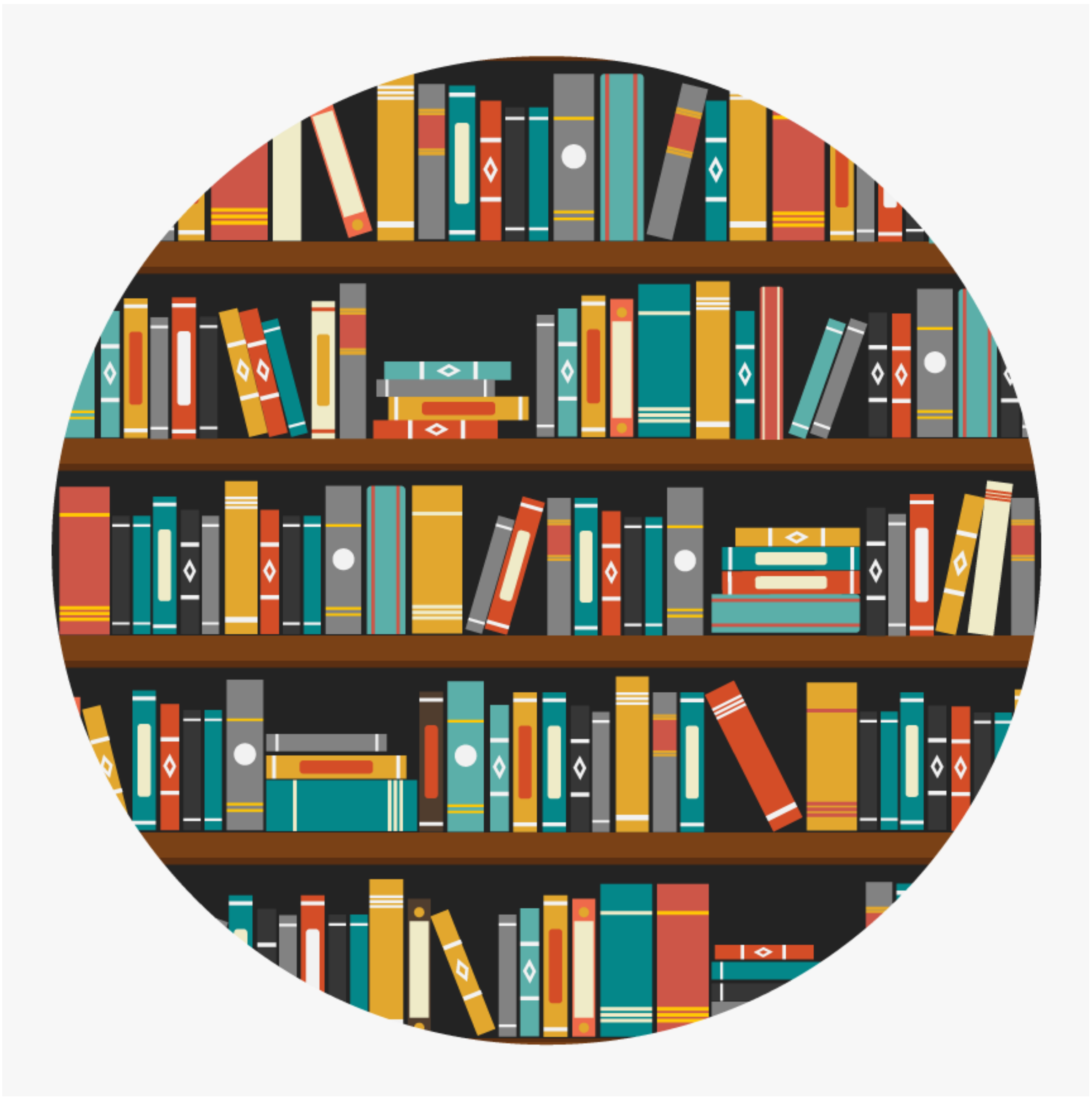
Book Finder Project



Participants

- Emil Latypov B20-03 (e.latypov@innopolis.university)
- Azamat Shakirov B20-03 (a.shakirov@innopolis.university)
- Bekzat Rakhimbayev B20-03 (b.rakhimbayev@innopolis.university)
- Shohjahon Khamrakulov B20-03 (s.khamrakulov@innopolis.university)
- Arlan Kuralbayev B20-03 (a.kuralbayev@innopolis.university)

Project Description

Book Finder was developed to organize and manage work of library.
The library system has Database of books and UI to interact with users.

Library Database stores list of books, which are represented as .csv file. After
each change the actual information is updating and saving in the file. Books contain name, author,
number of pages, category, year of publication, location and status. To make it easier to find
the book, the application provides the user with information about the shelf, row and column number,
where book is located. After borrowing the book becomes inaccessible to other users until
it is returned.

UserInterface is a wrapper, which provides to the user an opportunity to look for
books of interest to him by *book name* or *author name*, borrow available books and
return them and add a new ones by terminal.

Available features:

- Search any book by name or by author
- Get basic information about books such as name, author, category, realized year, page number etc.
- Keep track location of any book by just shelf, column and row number
- Easy to track book's status if book is borrowed by someone or not
- If book status is available, there is an option to borrow
- Return borrowed book by just entering its ID
- Add new book to database

How to use

Book Finder has user-friendly and easy to use User Interface on terminal. Limiting user control on Program by applying numbers for each command and it helps to use commands like real application buttons.
It is carefully ensured that user can only enter numbers that belong to available commands, program keeps giving notifications until user enters valid input.
For example:

```
-----------------------------------------
Welcome to Book Finder
-----------------------------------------
Please choose following commands:
0 -> Search
1 -> Return book
2 -> Add new book
3 -> Exit
Enter number >> 0

-----------------------------------------
SEARCHING!
-----------------------------------------
Please choose following commands:
0 -> Search by Name
1 -> Search by Author
2 -> Go Back
Enter number >> 0

-----------------------------------------
SEARCHING BY NAME
```

```
----------------------------------------
0 -> Go Back
Please choose commands or Search: agile
--------------------------------
FOUND RESULTS:
0 --> Agile Web Development with Rails BY: Sam Ruby, Dave Thomas, David
Heinemeier Hansson
1 --> Agile web development with rails: a Pragmatic guide BY: Dave Thomas, David
Heinemeier Hansson, Leon Breedt, Mike Clark, Thomas Fuchs, Andrea Schwarz
2 --> Agile.Web.Development.with.Rails.4th.Edition BY: the pragmatic programmers
Choose book from results
Enter number >> 0

----------------------------------------
BOOK DESCRIPTION
----------------------------------------
ID: 0
Name: Agile Web Development with Rails
Author: Sam Ruby, Dave Thomas, David Heinemeier Hansson
Category: Web development
Year: 2010
Pages: 472
Location -> shelf: 4 column: 3 row: 3
Status: Available
------------------------------------
0 -> Borrow book
1 -> Go Back
Enter number >> 0
You have successfully borrowed book with id = 0
----------------------------------------
Welcome to Book Finder
----------------------------------------
Please choose following commands:
0 -> Search
1 -> Return book
2 -> Add new book
3 -> Exit
Enter number >>
```

etc.

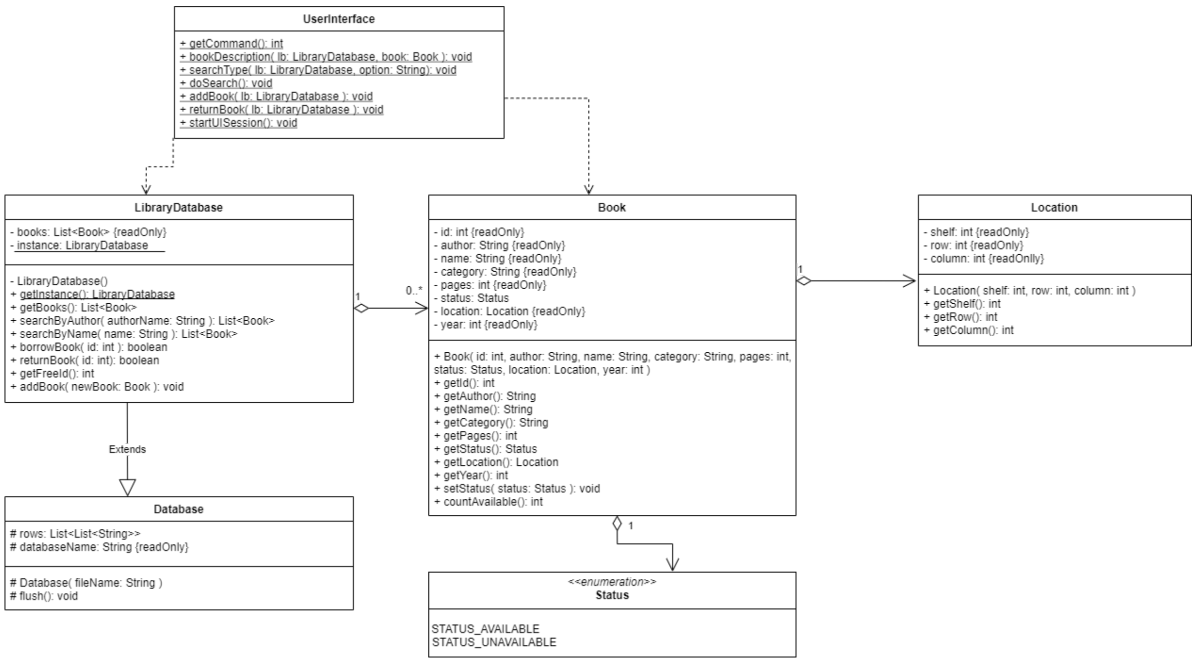
Project structure

At the beginning of the program, the control is forwarded into UserInterface's **startUISession** function. UserInterface is responsible for any communications with user, and each of its function helps with that. It provides laconic interface for user (which is described above). During it's work it makes requests to LibraryDatabase. It is single-instance class (implemented by **Singleton** pattern), which stores list of Books inside and provides methods to manipulate them and update database permanently (even in the disk). LibraryDatabase is inherited from Database class, which works with the database file (.csv format) in low level and provides high-level methods to manipulate data, doesn't matter which fields are stored in table. Main structure we are working with in out program is class Book. It stores all the data about book. Auxiliary class Location represents location of the book. Enum Status shows if book is free (*STATUS_AVAILABLE*) or borrowed (*STATUS_UNAVAILABLE*).

Why did we use Singleton?

Singleton pattern is used to design our Library Database. It should be ensured that from any place of our program we are working with the same instance of database - doesn't matter if it is called from UserInterface or Book class. In this task we used Classic Singleton implementation

UML


UserInterface
+ getCommand(): int
+ bookDescription(lb: LibraryDatabase, book: Book): void
+ searchType(lb: LibraryDatabase, option: String): void
+ doSearch(): void
+ addBook(lb: LibraryDatabase): void
+ returnBook(lb: LibraryDatabase): void
+ startUISession(): void
LibraryDatabase
- books: List<Book> {readOnly}
- instance: LibraryDatabase
- LibraryDatabase()
+ getInstance(): LibraryDatabase
+ getBooks(): List<Book>
+ searchByAuthor(authorName: String): List<Book>
+ searchByName(name: String): List<Book>
+ borrowBook(id: int): boolean
+ returnBook(id: int): boolean
+ getFreeId(): int
+ addBook(newBook: Book): void
1
0..*
Book
- id: int {readOnly}
- author: String {readOnly}
- name: String {readOnly}
- category: String {readOnly}
- pages: int {readOnly}
- status: Status
- location: Location {readOnly}
- year: int {readOnly}
+ Book(id: int, author: String, name: String, category: String, pages: int, status: Status, location: Location, year: int)
+ getId(): int
+ getAuthor(): String
+ getName(): String
+ getCategory(): String
+ getPages(): int
+ getStatus(): Status
+ getLocation(): Location
+ getYear(): int
+ setStatus(status: Status): void
+ countAvailable(): int
1
Location
- shelf: int {readOnly}
- row: int {readOnly}
- column: int {readOnlly}
+ Location(shelf: int, row: int, column: int)
+ getShelf(): int
+ getRow(): int
+ getColumn(): int
Extends
Database
# rows: List<List<String>>
# databaseName: String {readOnly}
# Database(fileName: String)
# flush(): void
1
<<enumeration>>
Status
STATUS_AVAILABLE
STATUS_UNAVAILABLE