Online Bookstore System “Booka”

- ***What is “Booka”?***

  The online bookstore "Booka" is a web service that allows users to rent books. To do this, the user must fill out a booking form. Also, the service generates reports and provides an opportunity to see which books are available and which of them are already booked. The user can select books by alphabet, genre and author.

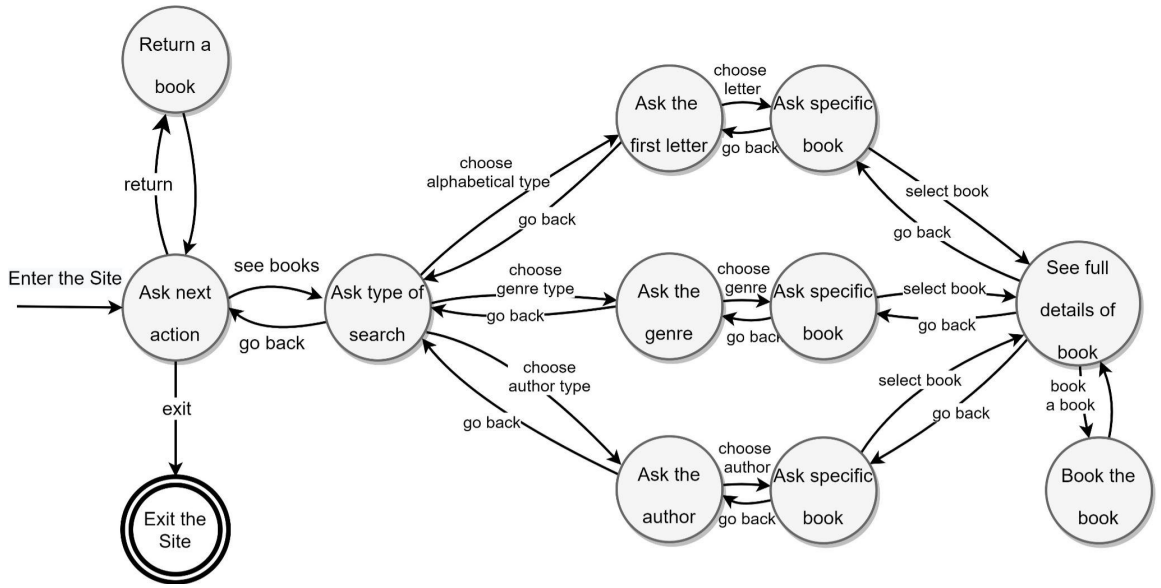
- ***Aim of the project***

  The goal of the project is to create a book rental web service that would be ***user-friendly, reliable & efficient*** in terms of using software resources. We should ***use programming design patterns and OOP principles*** for this aim.

- ***What has been done at this stage of development***

  The original ***structure of the project*** has been changed, ***structural design patterns have been implemented,*** such as Bridge and Flyweight, that modernize the existing project, ***new functionality*** has been added, such as author search, history preservation. The ***process of adding new books has been improved*** and the ***process of adding new functionality*** for the user has been ***facilitated***.

- ***State machine diagram***
- To understand how the user interacts with the web service, a ***state machine diagram*** was built that describes in detail the user's capabilities. The **State** pattern was used to implement this behavior.



- ***User interaction with the system***

  The user interacts with the system only through the console (there is no need to change anything manually in the code). The actions that the user can perform are written on the screen.

```
Welcome to the "Booka" Bookstore website.
What do you want to do?
{(0)RETURN_BOOK; (1)SEE_BOOKS; (-)EXIT}
1

~~~~~~~~~~~~~~~~~~~~~~
Which type of search do you prefer?
{(0)ALPHABETIC; (1)GENRE; (2)AUTHOR; (-)EXIT}
1

~~~~~~~~~~~~~~~~~~~~~~
Drama (4 books)
Romance (6 books)
Detective (3 books)
Horror (3 books)
Psychological (2 books)
Fiction (9 books)
Comedy (2 books)
Erotic (5 books)
Thriller (1 books)
Ecchi (1 books)
Type a 'genre' to find?
{(-)EXIT}
Drama

~~~~~~~~~~~~~~~~~~~~~~
Drama:
        War and Peace {Nikolay Tolstoy} id=10
        Spice and Wolf {Isuna Hasekura} id=31
        Anna Karenina {Nikolay Tolstoy} id=11
        Mumu {Nikolay Tolstoy} id=9

Choose book by ID {(-)EXIT}
10

~~~~~~~~~~~~~~~~~~~~~~
book: War and Peace
genre: Drama
author: Nikolay Tolstoy
price: 309.8$
This book is already reserved.
{(-)EXIT}
```

```
~~~~~~~~~~~~~~~~~~~~~~
Welcome to the "Booka" Bookstore website.
What do you want to do?
{(0)RETURN_BOOK; (1)SEE_BOOKS; (-)EXIT}
1

~~~~~~~~~~~~~~~~~~~~~~
Which type of search do you prefer?
{(0)ALPHABETIC; (1)GENRE; (2)AUTHOR; (-)EXIT}
0

~~~~~~~~~~~~~~~~~~~~~~
A (1 books)
B (1 books)
C (2 books)
D (1 books)
F (1 books)
H (1 books)
I (1 books)
J (1 books)
K (1 books)
L (2 books)
M (5 books)
N (2 books)
P (2 books)
R (3 books)
S (2 books)
T (7 books)
V (1 books)
W (1 books)
Y (1 books)
Type a letter to find?
{(-)EXIT}
R

~~~~~~~~~~~~~~~~~~~~~~
R:
        Ronaldo {Thinker Cristiano} id=8
        Re:Zero - Starting Life in Another World {NAGATSUKI Tappei} id=25
        Rust {Alexsander Krug} id=3

Choose book by ID {(-)EXIT}
```

book booking user interaction and book returning user interaction

```
Thriller (1 books)
Ecchi (1 books)
Type a 'genre' to find?
{(-)EXIT}
Psychological

~~~~~~~~~~~~~~~~~~~~~~
Psychological:
        I'm A Spider, So What? {BABA Okina} id=27
        How to forget C++, for dummies {Rifujin Magonote} id=30

Choose book by ID {(-)EXIT}
27

~~~~~~~~~~~~~~~~~~~~~~
book: I'm A Spider, So What?
genre: Psychological
author: BABA Okina
price: 61.5$
Want to reserve a book?
{(0)NO; (1)YES; (-)EXIT}
1

~~~~~~~~~~~~~~~~~~~~~~
Enter your first name {(-)EXIT}
Artem
Enter your last name {(-)EXIT}
Voronov
To book a book, you will need to pay money, do you agree?
{(0)NO; (1)YES; (-)EXIT}
1
The book is successfully reserved (Please remember book ID = 27).
{Press enter to continue}


~~~~~~~~~~~~~~~~~~~~~~
book: I'm A Spider, So What?
genre: Psychological
author: BABA Okina
price: 61.5$
This book is already reserved.
{(-)EXIT}
```

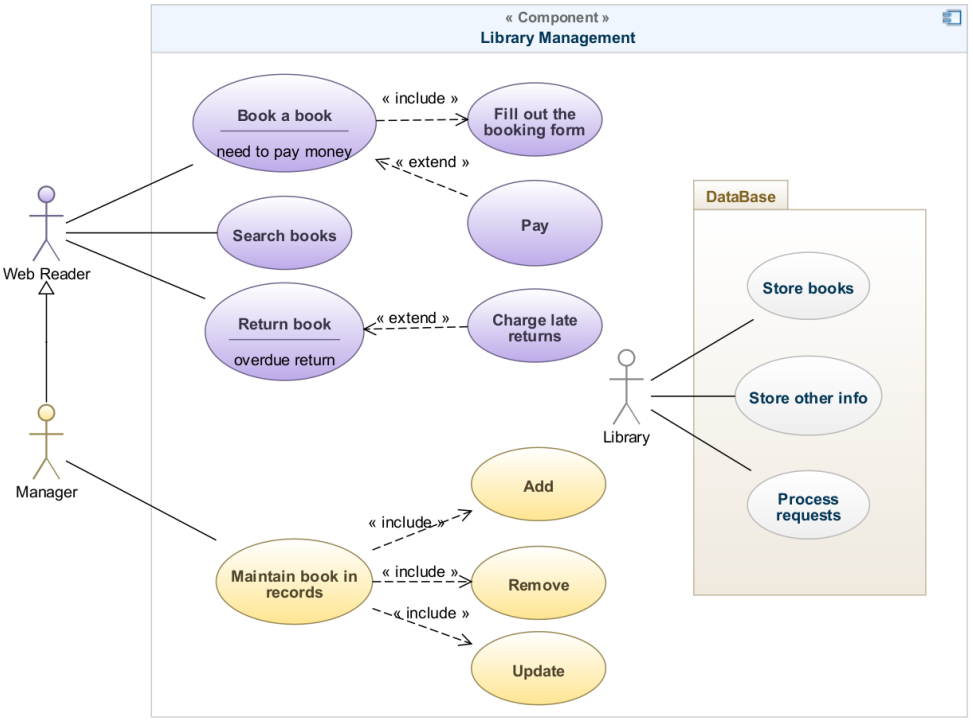
```
~~~~~~~~~~~~~~~~~~~~~~
Welcome to the "Booka" Bookstore website.
What do you want to do?
{(0)RETURN_BOOK; (1)SEE_BOOKS; (-)EXIT}
0

~~~~~~~~~~~~~~~~~~~~~~
Enter your first name {(-)EXIT}
Artem
Enter your last name {(-)EXIT}
Voronov
Ordered books:
0)      I'm A Spider, So What? {BABA Okina} id=27
Enter the order number you want to return or {(-)EXIT}
0
You successfully returned the book.
{Press enter to continue}


~~~~~~~~~~~~~~~~~~~~~~
Welcome to the "Booka" Bookstore website.
What do you want to do?
{(0)RETURN_BOOK; (1)SEE_BOOKS; (-)EXIT}
```

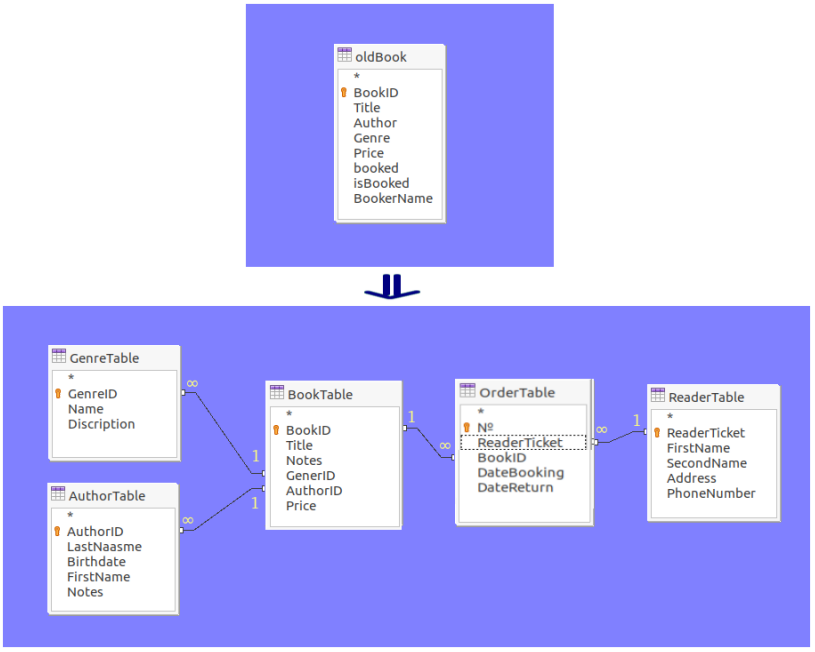
- ***Statechart diagram***

  To understand how the user interacts with the web service, a ***statechart diagram*** was built.



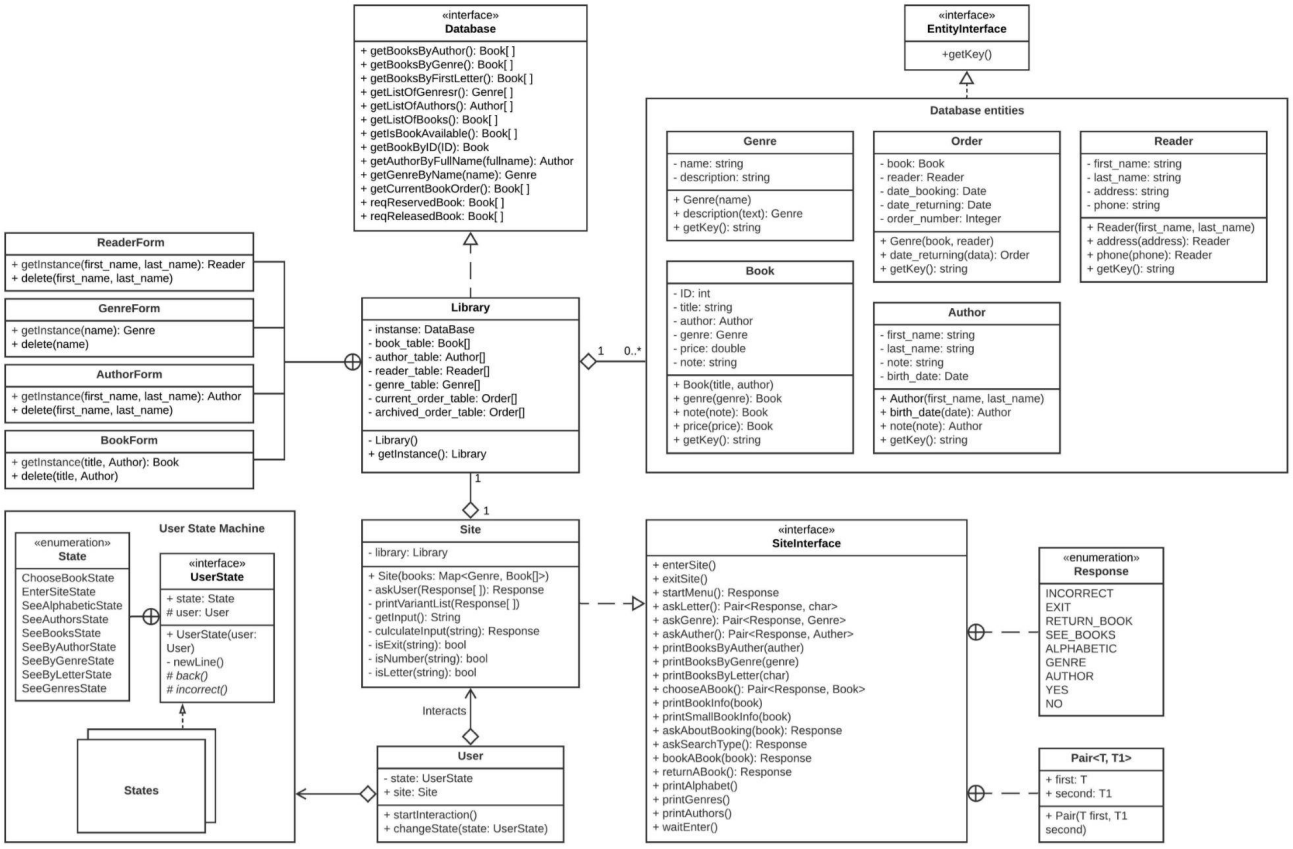
- ***Changes in system structure***

  Previously, the project consisted of one large database, with which the site interface interacted. The disadvantage of this approach is the duplication of identical elements, such as the same author names. When expanding the functionality of the library, the class gets too bloated, so it becomes difficult to maintain and change it. To avoid the above disadvantages, it was decided to spread the data of one database into several data tables which store general information. This strategy allows adding and removing entities more flexibly, saves the memory used by the application, and facilitates further modernization of the project. The Bridge and Lightweight patterns were used to implement these structural changes.



- ***Current UML class diagram***

When creating a new UML class diagram, the old existing diagram was initially analyzed. When choosing design patterns and a new project structure, we started from the disadvantages found in the old version, trying to eliminate them, as well as generally improve the functionality and simplify work with the project.



- ***All design patterns used in the project:***

➔ ***Singleton*** (from the 1st stage of development);

➔ ***Bridge*** (code with the separated interface and its implementation);

➔ ***Builder*** (for convenient creation of database entities);

➔ ***Flyweight*** (reorganization of DataBase, reduce the amount of memory used and prevent duplication of identical entities, create/modify/delete an entity, provides access Point (Form)); - main design pattern

- ***State*** (for user interaction with the site)

- ***Flyweight as the main design pattern on this iteration of project development***

Briefly about this design pattern:

Flyweight is a structural design pattern that allows you to fit more objects into the allocated RAM. Flyweight saves memory by sharing the general state of objects among themselves, instead of storing the same data in each object.

Why was Flyweight chosen for our project?

As already mentioned, earlier we had a lot of duplicate entities in the books, which led to duplication. Therefore, the main reason for choosing the lightweight pattern is the desire to reduce memory consumption by transferring similar data to a separate class. Example - each book stored its author, but since one author can have many books, which means the data was duplicated. Instead, now each book refers to the author, and does not store it as its essence. To do this, the first thing we did was reorganize the database, putting duplicate data into separate tables. (see the section "changes in system structure") The structure of Java packages repeats each other in the implementation branch and interfaces.

- ***Examples of code snippets with pattern implementation***

➢ **Singleton pattern**

Has not changed since the last iteration. Gives shared access to the database from different parts of the program and guarantees that there’s just one instance of a class.

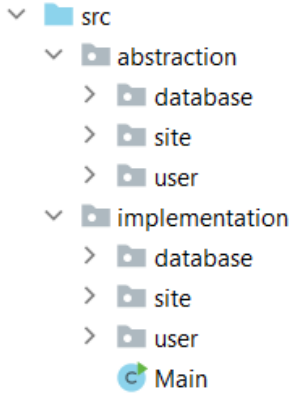
```java
/**
 * An alternative to the constructor and is the access point to an instance of
 * Library class.
 */
public static Library getInstance() {
    if (instance == null)
        instance = new Library();
    return instance;
}

public final class Library implements DataBaseInterface {

    /** Singleton object */
    private static Library instance;
```

➢ Bridge pattern

The Bridge pattern is the separation of code into abstraction and implementation. The implementation of this pattern is visible in the project architecture (abstraction and implementation folders)



➢ Builder pattern

In the builder, those fields that are not mandatory for the constructor were taken out. E. g., using the builder, one can add a description to the genre.

```java
public Genre description(String description) {
    this.description = description;
    return this;
}
```

```java
public class Genre implements EntityInterface {
    /**
     * Property of Genre
     *
     * final means key field
     */
    private final String name;
    private String description;
```

➢ Flyweight pattern

Implementation of the Flyweight pattern on the example of genres

```java
/**
 * This class is access point for creating, modifying, and deleting a Genre.
 * Prevents duplication.
 *
 * TODO configuring administrator-only access rights
 */
public class GenreForm {
    public static Genre getInstance(String name) {
        String _genre_key = Genre.getKey(name);
        if (null == genre_table.get(_genre_key))
            genre_table.put(_genre_key, new Genre(name));

        return genre_table.get(_genre_key);
    }

    public static void delete(String name) {
        String _genre_key = Genre.getKey(name);
        genre_table.remove(_genre_key);
    }
}
```

➢ **State pattern**

This pattern makes it possible to effectively implement a state machine that can be modified without difficulty.


```
public abstract class UserInterface {
    private UserState state;
    public Site site;

    public UserInterface(Site site){
        this.site = site;
        state = null;
    }
```

- ***Conclusion***

At this stage of development, we have changed the structure of the project, improved the old functionality, and introduced a new one. The use of design patterns allowed to modernize the system and facilitate further expansion of the project in the future. Thus, the goals of the project were realized.