Library Project Summary

The library project consists of two parts:

1. The executable **"Main"**, which uses a command-line interface.
2. The static library **"OOPLib"**.

The executable **"Main"** utilizes **"OOPLib"** to execute the library project.

The executable “Main”:

The executable **"Main"** includes the project's main function. The main function utilizes two key functions: **user verification** and **launching the app**.

User Verification

This function verifies the identity of the user launching the app.

- First, an option to restore a forgotten password is provided, using RSA encoding for security (explained later).
- Then, the function verifies the user's ID and password.

Launch the App

This function manages the library's interface.

- It presents the available actions the user can take after logging into the app.

The Main Function Workflow

1. Creates an instance of the library.
2. Calls the **User Verification** function.
3. Calls the **Launch the App** function.

The static library “OOPLib” :

The static library **"OOPLib"** includes ten classes for managing the library:
Library, User, Librarian, Loaner, Reservation, Loan, LoanHistory, Book, RSA, and Exception.

It interacts with the following CSV files to load and store data when the program starts and after it terminates:

- **Books.csv**

- **Librarians.csv**
- **Loaners.csv**
- For each loaner with ID **X**:
    - **X_loans.csv**
    - **X_reservations.csv**
    - **X_loanHistory.csv**

Class Exception:

This class centralizes various self-defined exceptions, which are thrown when an issue arises while using the app. These include:

- **Invalid command**
- **Invalid verification**
- **Forbidden action**
- **Unrecognized action**
- And other error cases
-

Class RSA:

This class implements an RSA encoding mechanism, which is used in the following cases:

- When a user forgets their password.
- For the **"reserve secretly"** function, allowing a loaner to reserve a book discreetly and prevent competition for high-demand books.

RSA encoding scheme:

1.Library chooses to big prime numbers, p,q.

2.Library find a number e, such that gcd(e,(p-1)(q-1))=1, and define n=p*q.

3.Library find the inverse of e modulo (p-1)(q-1) , $d = e^{-1} mod((p-1)(q-1))$

4.Library defines: $E: z_n^* \to z_n^*$, $E(x) = x^e (mod n), and\ publishes\ (n, e) to\ the\ users.$

5.B sends a message x to the library by sending E(x).

6.Library deciphers y=E(x) by using $D: z_n^* \to z_n^*$, $D(y) = y^d (mod n)$

7. $DE(x) = x^{e*d} = x(mod n)$

The class utilizes several functions: **isPrime, gcd, random_prime, find_encoder, modInverse, and modExponentiation**, following the scheme described above.

It stores **n, e, and d** within the class (**but not p and q!**).

Additionally, it provides public methods that:

- Return **n** and **e** for the user.
- Use the **D function** to perform decryption for the library

Class book:

This class stores all information about the books in the library, corresponding to **books.csv**.

It maintains the following details for each book:

- **ISBN**
- **Title**
- **Author**
- **Year of publication**
- **Category**
- **Quantity available**
- **Maximum loan duration (in days)**

Classes loan and loanHistory:

These classes store all information about a loaner's current and past loans, corresponding to the loaner's CSV files.

Both classes maintain the following details for each loan:

- **Loan date**
- **Return date**
- **Book ISBN**

Class reservation:

This class stores all information about a loaner's reservations.

It maintains the following details for each reservation:

- **Reservation date**
- **Book ISBN**
- **days_from_2000** parameter, initialized using the **days_from_2000** function, to track how long a book has been reserved (explained later).

Class user:

This class stores all information about the users of the application. It is an abstract class, as there are two types of users: **loaners** and **librarians**, both of whom share common properties. To accommodate this, two subclasses—**Loaner** and **Librarian**—inherit properties from the **User** class.

It stores the following properties:

- **ID**
- **Name**
- **Password**

The **User** class includes three pure virtual functions:

1. **operator<<**
2. **prepare**
3. **search_info**

The **prepare** function's purpose is to perform actions automatically once the user enters the app (explained later).
The **search_info** function facilitates a search action, suggesting options for the user.

Class librarian:

This class stores all information about the librarians in the library. It inherits the following properties from the **User** class:

- **ID**
- **Name**
- **Password**

It implements the following virtual functions:

1. **operator<<**
2. **prepare**
3. **search_info**

Prepare Function

This function calls the **prepare** function for all loaners.

Search_info Function

This function searches for loaners based on the name provided as an argument or retrieves the book history of each loaner.

Class loaner:

This class stores all information about the loaners in the library. It inherits the following properties from the **User** class:

- **ID**
- **Name**
- **Password**

It implements the following virtual functions:

1. **operator<<**
2. **prepare**
3. **search_info**

The class contains the following data members:

- **loan_history** (vector)
- **loans** (vector)
- **reservations** (vector)
- **max_loans** (parameter)

Additionally, the **Librarian** class is declared as a friend class for the **prepare** and **search_info** methods.

Constructor

The constructor interacts with the CSV files to load data into the program.

Prepare Function

When a loaner enters the app, the **prepare** function is called. It performs the following actions:

- First, the function checks if there are any expired loans.
- Then, it checks for reservations that have been held for more than a week and deletes them from their corresponding vectors.

reservations_to_loans Function

After the **prepare** function, the **reservations_to_loans** function is called. This function suggests that the loaner can loan books from the reservation vector if the books are in stock and the loaner has not exceeded the maximum number of allowed loans.

Search_info Function

This function searches for the book history associated with the loaner.

Destructor

Finally, the destructor uploads any data modified by the app back to the CSV files.

Class library:

The purpose of this class is to manage the library.

It stores the following:

- **Current date**
- **Vectors of users and books**
- **Set of book categories**
- **MaxLoans static variable**
- **RSA instantiation** for encoded actions

The **Librarian** class is a friend class of the **Library** for the **prepare** and **search_info** functions.

Constructor

The constructor interacts with the CSV files to load data into the program. It also initializes the current date and calls the **loans_and_reservations_preparation** function, which in turn calls the **prepare** method for all users.

User Actions

The main function suggests the following actions for every user:

- **Print <ISBN>**

- **PrintBooks <category>**
- **More**
- **Search <search_text>**
- **Update**

For **Librarian**, additional actions are suggested:

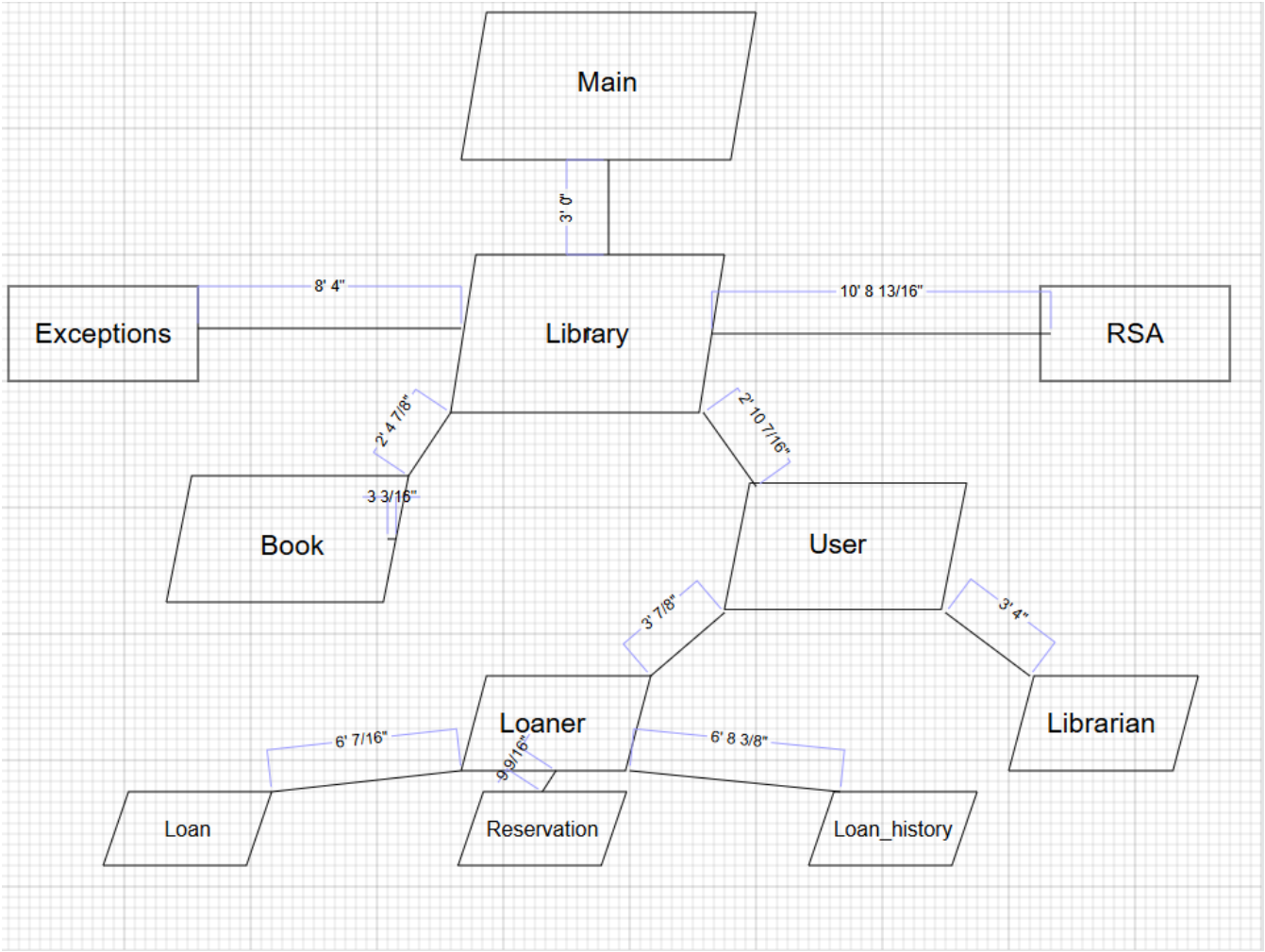
- **AddBook**
- **UpdateBook**
- **RmBook <ISBN>**
- **AddClient**
- **RmClient <id>**
- **AddCategory <category>**
- **RmCategory <category>**
- **setMaxLoans <numLoans>**

For **Loaner**, the following actions are suggested:

- **Loan <ISBN>**
- **Loan <name>**
- **Return <ISBN>**
- **Return <name>**
- **Cancel <ISBN>**
- **Cancel <name>**
- **PrintLoans**
- **PrintReservation**
- **PrintHistory**

Destructor

Finally, the destructor uploads any data modified by the app back to the CSV files.


Main
3' 0"
Exceptions
8' 4"
Library
10' 8 13/16"
RSA
2' 4 7/8"
2' 10 7/16"
3 3/16"
Book
User
3' 7/8"
3' 4"
Loaner
Librarian
6' 7/16"
9 9/16"
6' 8 3/8"
Loan
Reservation
Loan_history