The Library class as used for the Cofoja assignment has been modified for this task. There are three functions that are relevant for the assignment:

1. find
2. addMember
3. borrowbook

The task is to add required JML specifications to the methods and use “KeY” to prove correctness of the methods. We need not change the source code for the task.

The “find” function requires a loop invariant. Please refer to SumAndMax example of KeY: https://courses.cs.ttu.ee/w/images/2/2a/ITI0130_Lab11_summax.zip

The requirements for the contracts are provided in the comments. Please implement them in JML. Some trivial specifications (e.g. assignable) are already provided.

If the tool fails to prove with the given contracts, please explore the proof tree and open goals to check if any additional contract can successfully prove.

The configuration for proof strategy is:

- Stop at: Default
- Proof splitting: Delayed
- Loop treatment: Invariant
- Block treatment: Contract
- Method treatment: Contract
- Dependency contracts: On
- Query treatment: Off
- Expand local queries: On
- Arithmetic treatment: Basic
- Quantifier treatment: No Splits with Progs
- Auto Induction: Off
- User specific taclet sets: <all off>

Please set the Max. Rule applications to at least 10K. If the proof requires more rules, you may have to press the proof button multiple times, till it completes the proof (show statistics) or display the message “No rules applied” on the status bar below.

Some JML notations, which may be of help for the task, are discussed below.

- \forall
- \exists
- ==>
- \old
- \result
- Decreases

```
(\forall int k; 0 <= k && k < bookISBN.length; bookISBN[k] != 0)
```

The above formula states that for all values of variable k, in the range 0<=k<bookISBN.length, must satisfy the property “bookISBN[k] != 0”. The expression will evaluate to false if there is any k for which the property is false.

```
(\exists int k; 0 <= k && k < bookISBN.length; bookISBN[k] == isbn)
```

\exists as in the formula above states that “there exists” a value of k in the range 0<=k<bookISBN.length, such that the property “bookISBN[k] == isbn” is true. The formula will evaluate to false if there is no such k for which the property is true.

```
(\forall int k; 0 <= k && k < bookISBN.length; bookISBN[k] != 0)
      ==> (\exists int k; 0 <= k && k < bookISBN.length; bookISBN[k] == isbn);
```

The sign “==>“used in the above formula denotes implication in logic. The above formula states that if the array bookISBN[] does not contain any element with value of 0, then there must exist an element with value isbn in the same array.

```
bookISBN[k] == \old(isbn)
```

\old refers to the value of variable in the pre- state. This is used in post-condition to refer to values of variables in the pre-condition state. The above formula states the equality of bookISBN[k] and the old value of isbn.

```
\result < a.length ==> a[\result] == key
```

\result refers to the return value of a method.

```
decreases (x - i);
```

The decreases keyword is used to specify loop variants.

The \forall and \exists can be nested to for more complex logic as shown:

```
(\exists int k; 0 <= k && k < memberID.length; memberID[k] == \old(mid) &&
borrowed[k] && (\forall int m; 0 <= m && m < borrowed.length && m != k;
\old(borrowed[m]) == borrowed[m]));
```