The​ ​Hong​ ​Kong​ ​Polytechnic​ ​University

Department​ ​of​ ​Electronic​ ​and​ ​Information​ ​Engineering

EIE3320​

​Object-Oriented Design​ ​and​ ​Programming

Lab 2 report: Library Admin System

**Introduction**

**Program requirements and objectives**

The following application is a library administration system that allows the user to display, add, edit, delete, save current data, load stored data, load test data, sort and search book records. In addition, the application allows the user to borrow/ return books, to add/show images for each book and show the waiting queue for a particular book. The user interface (GUI) is created with Java Swing which validates user actions with error messages and allows the user to easily interact with the system. The user should be capable of closing the application by clicking the “Exit” button or the “X” button on the top right corner of the application window.

In order to meet the program requirements above following objectives have been set:

1. Create a custom linked list class “MyLinkedList” that implements the custom list interface “MyList”. MyLinkedList class contains all functions required by a normal linked list
2. Create a custom queue class “MyQueue” that makes use of MyLinkedList class and contains basic methods required for a queue.
3. Test the implementation of both MyLinkedList and MyQueue classes with given test classes and check the output
4. Design the UML diagram of the whole system
5. Create a “Book” class that is used to store information about a particular book, including the ISBN, title, availability, waiting queue and the image.
6. Test the implementation of Book class with given test class and check the output
7. Create a GUI of the application by using Java Swing. The GUI of the program consists of the main window(JFrame) and pop-up windows(JDialog).
   1. Create a Frame class that contains the main UI for the user with JTextArea to show user name, JTable to show book records and JPanel with JButtons and JTextFields for the user interaction
   2. Create a custom Dialog class for the pop-up window that shows additional information about a particular book.
8. Implement all of the required features for the application
9. Test the program

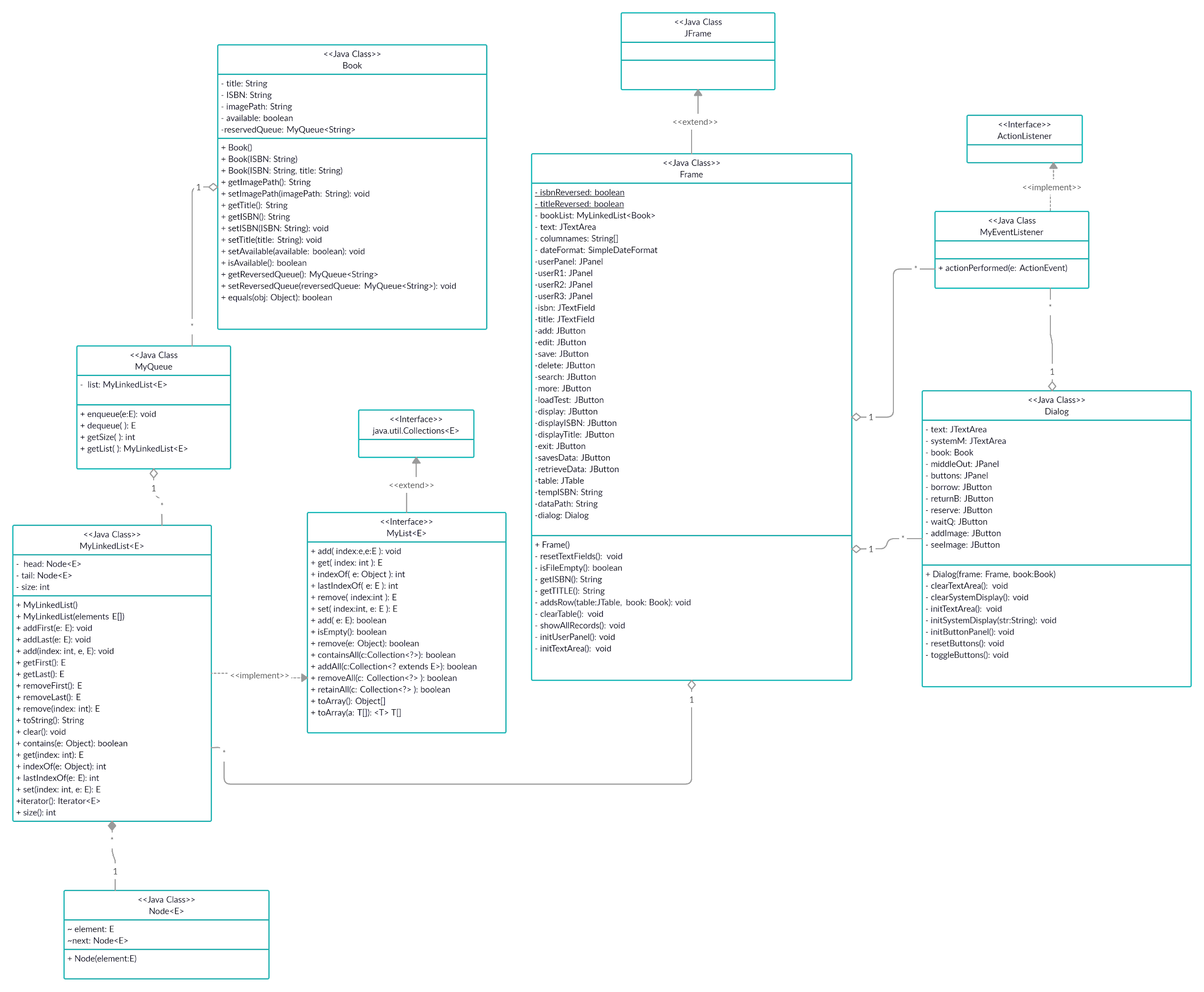
**Methodology**

I was working alone for this laboratory, so both code and report were prepared by me.

**Schedule**

|  |  |
| --- | --- |
| **Day No.** | **Activity** |
| 1 | 1. Analysis of MyList interface 2. Creation of MyLinkedList and MyQueue classes 3. Testing of MyLinkedList and MyQueue classes 4. Creation of Book class 5. Testing of Book class 6. Creation of UML diagram 7. Study of JTable, JOptionPane and JDialog 8. Creation of GUI of the main frame |
| 2 | 1. Development of add, load test data, edit&save, delete,display all, search, display by isbn, display by title, exit features 2. Testing and debugging of features described above 3. Development and testing of “more” feature    1. Creation of custom Dialog class 4. Test all developed features |
| 3 | 1. Develop and test add/show image feature 2. Develop and test save/retrieve data feature 3. Code refactoring 4. Preparation of code for submission |
| 4 | 1. Preparation of laboratory report |
| 5 | 1. Video recording for demonstration 2. Submission |

**UML Class Diagram**

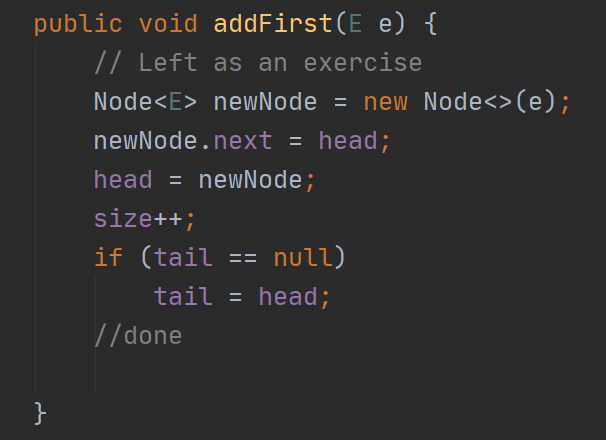
****

**Implementation of MyLinkedList**

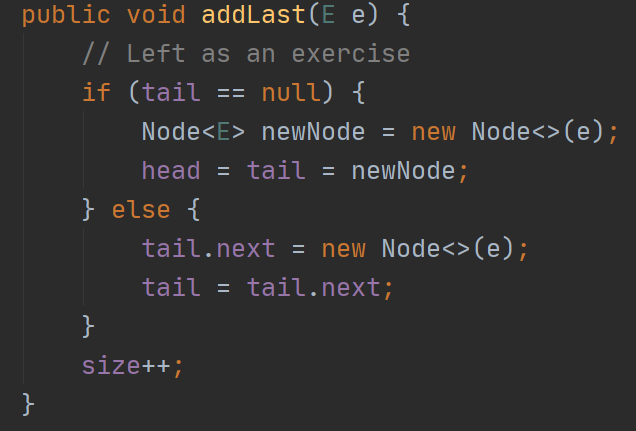
MyLinkedList is a custom class that implements a custom MyList interface. MyList interface extends Collection interface. In MyList there are abstract methods which are implemented in MyLinkedList class and some methods are already realized as default methods. MyLinkedList consists and operates with nodes of generic type.

Node class has 2 variables : element which stores the data of the node and variable next of Node<E> type which points to the next node. The constructor of Node class initializes the variables.

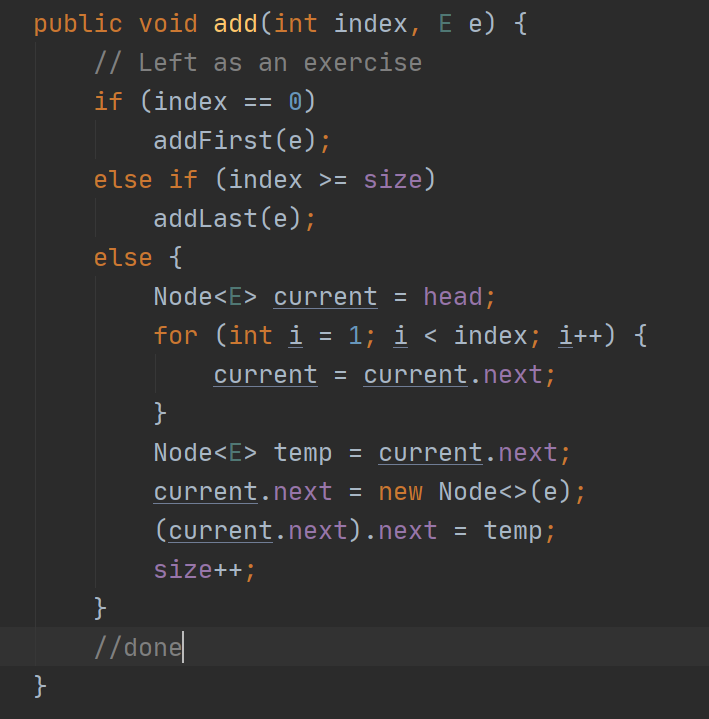
Below is a brief explanation of important methods of MyLinkedList

1. Major function of any linked list is add operation. In MyLinkedList there are different types of add operations:  
   

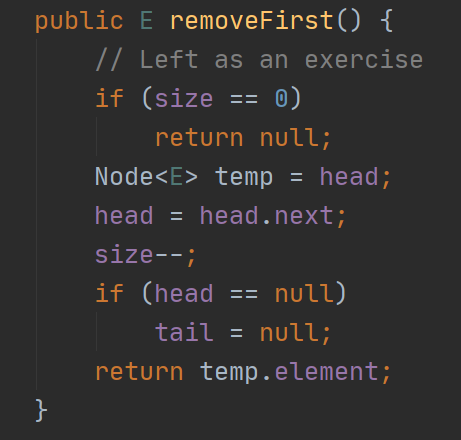
addFirst(E e) takes a variable of generic type as a parameter,creates a corresponding node with a variable specified as an element of a node. Then, node`s next pointer is set to head while the head now points to the newly created node. Lastly, we check if the linked list is empty, the tail would point to null and in this case if a new node is added tail should point to the same node as head.



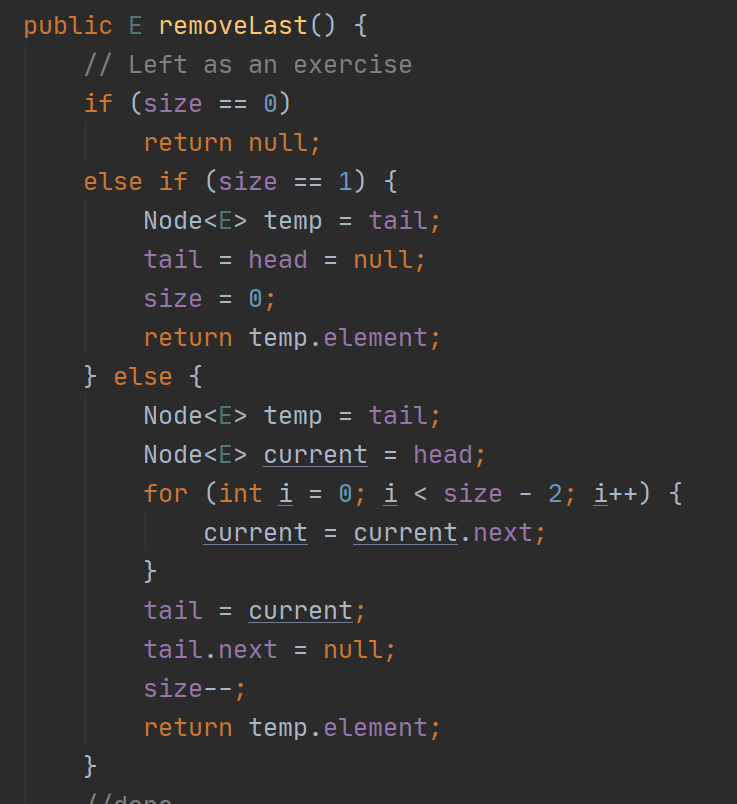
Similar approach is used here - a new node is created, but this time we first check if the linked list is empty in that case we assign both head and tail to point to that node. Otherwise, we assign the next value of tail to point to new node and then set tail to point to the new node.



For this method, we make use of previously created methods if the object is added to the beginning or the end of the linked list. Otherwise, we traverse the linked list until the desired position. We store the node at selected position to a temporary node variable and then reassign the node at current position to be our newly created node. Then, we set the next value of the current node (new node) to point to an old node that is stored in a temporary variable. Lastly, we increase the size.

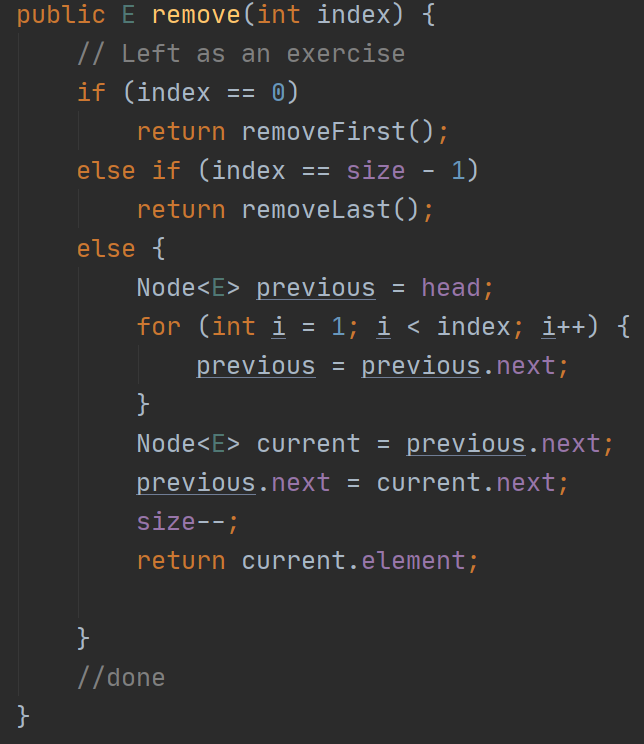
1. The next major operation of linked list is remove.  
   

RemoveFirst, firstly checks whether the list is empty and in this case returns null. Otherwise,we store the node in a temp variable that was stored in head. Reassign the head to point to the next node,decrement the size. Then, we check if the linked list is now empty, if so both head and tail now are null.Lastly, we return the removed object that was stored in a removed node (using temp node variable).

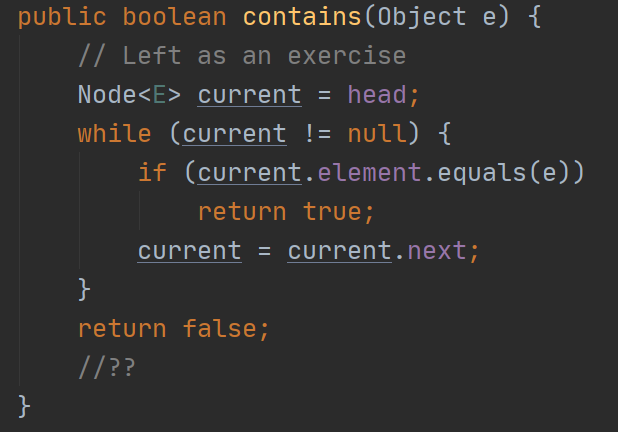


Same approach is used, we check if the linked list is empty and return null. If its size is 1, we store either tail or head (tail in my code) in a temp node variable. assign both tail and head to null and set the size to 0. Lastly, we return the removed element which is stored in a temporary node.

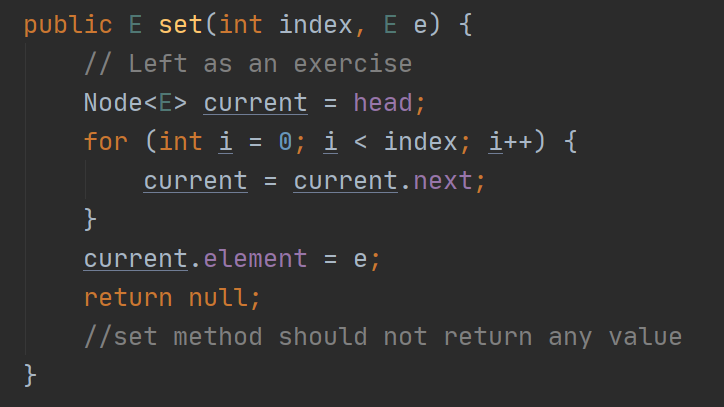
Otherwise if the list is neither empty nor its size equals 1, the node pointed by tail is stored in a temporary variable. Then we traverse to the penultimate node using a loop, set the tail to point to that node and set the next of the tail to store null. Lastly, we decrement the size and return the removed element.

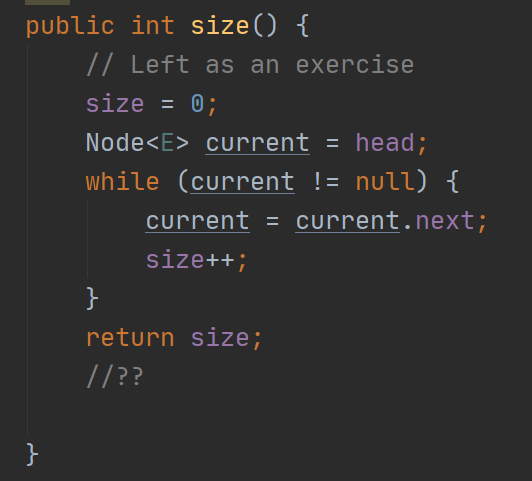


For this method we make use of previously created methods to remove the first/last element from the linked list. If the element is neither first nor last, we traverse to the node that is before the “node to be removed”.Then the “node to be removed” is stored in a temporary node variable. Then we set the next of the previous node to point to the node which is stored in the next of the “node to be removed”. Lastly we decrement the size and return the removed element. Basic idea behind all types of remove operation in a linked list is about deleting all references to this node. After some time this node will be deleted by the garbage collector.

1. Contains is another important method of a linked list  
   

Its implementation is simple. We traverse through the linked list and check every node`s element with the desired element. True and false are returned accordingly

1. The set method, looks for a desired node and simply replaces the element of this node to the desired one  
   
2. The size method is frequently used in the program. Its implementation is simple, the size variable is resetted and then we traverse through the linked list and count the number of nodes and hence size is updated.



At this point I understand that other operations already update the size variable so the size method could just return the size variable. However, I think this implementation is more safe.

**Usage of MyLinkedList in the program**

In the program, MyLinkedList is used first of all to store the object of the Book class.(typo of objects stored is Book as indicated in the angle brackets)



The booklist is then frequently used in the majority of basic and additional features. For example, we check whether booklist already contains the particular book before adding it in add and load test features:

if (!bookList.contains(temp)) {

temp.setTitle(getTITLE());

bookList.add(temp);

addsRow(table, temp);

} else if (bookList.contains(temp)) {

JOptionPane.*showMessageDialog*(Frame.this, "java.lang.Exception: Error: Book already exists");

resetTextFields();

return;

}

Whenever add/load test operations are performed and successful, a new book object is added to the booklist.

In addition, we also check whether the booklist is empty when the program is just loaded to prevent the user from operating on the system while no books are added.

For the delete operation we check if the booklist is not empty and whether booklist contains book with specified isbn and then proceed on deleting the book from the booklist:

Book temp = new Book();

temp.setISBN(getISBN());

if (!bookList.isEmpty() && bookList.contains(temp)) {

int tempIndex = bookList.indexOf(temp);

bookList.remove(tempIndex);

((DefaultTableModel) table.getModel()).removeRow(tempIndex);

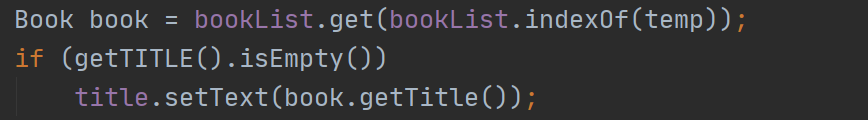
resetTextFields();

}else {

JOptionPane.*showMessageDialog*(Frame.this, "Error: book ISBN is not in the database");

}

Same checking is performed for the edit operation. In addition to that in edit operation, if only isbn was set we look for the book in the booklist according to its isbn and then set the text title text field to show the title of the corresponding book.



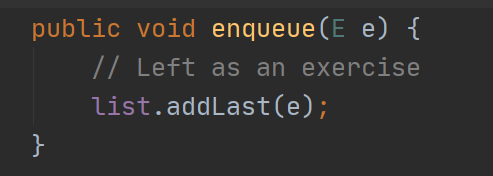
The next usage of the MyLinkedList is in the implementation of the MyQueue class.

**Implementation of the MyQueue class**

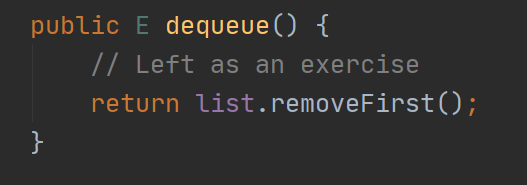
MyQueue class is implemented solely with MyLinkedList class.



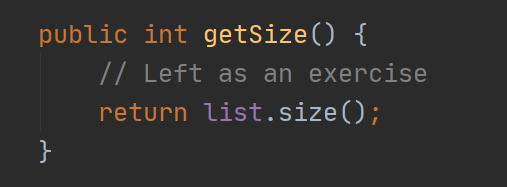
Its enqueue method adds new objects at the end of the linked list



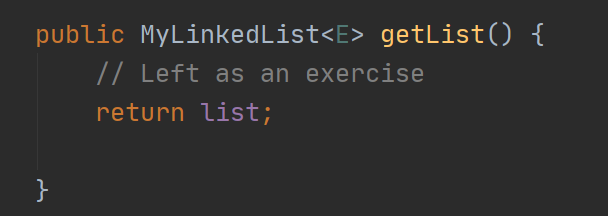
The dequeue method removes the first element of the linked list and also returns the removed value



Time complexity of addLast and removeFirst operations for a linked list is O(1) that's why linked list is favorable for queue implementation



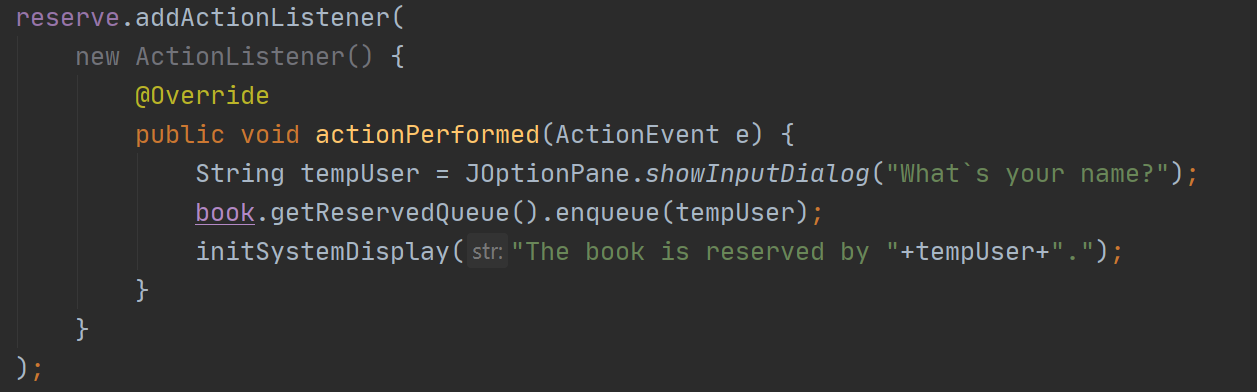
The getSize simply returns the size of the linked list used.



The getList returns the list

**The usage of MyQueue in the program**

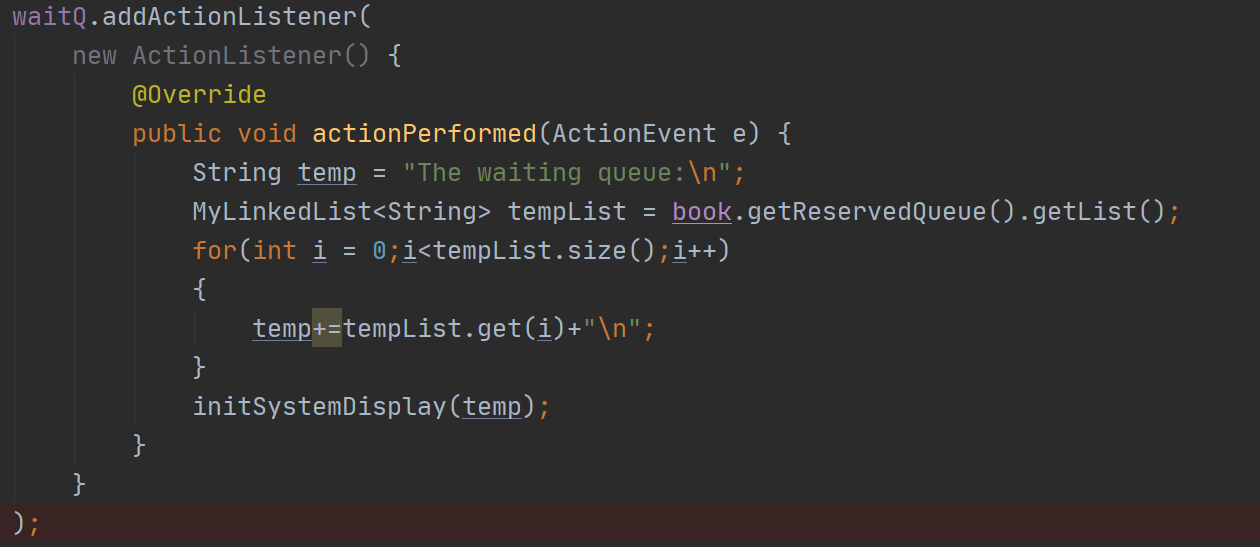
In the Library system each book can be borrowed and MyQueue is used to store the waiting queue of the people who reserved it. Queue operation is used when new person reserved it and dequeue is used when the person who currently owns a book has returned it:



Each time we want to access the waiting queue of the particular book we have to call getReservedQueue() .For return operation:



Finally, if we want to display the waiting queue of a particular book we need to traverse it as a linked list



Code sections above show how queue is used in the program.

**Additional features**

**Image Retrieval**

The feature is available once “more” buttons is pressed and a dialog menu for additional information about the book appears. This feature consists of 2 actions : adding and showing the image.

For adding the image I have created a JButton name addImage and added an anonymous action listener to it.



When the button is pressed, the file chooser pops up and allows the user to select the image file. Files seen by the user are filtered with FileNameExtensionFilter, so only images are seen. Once an image is selected and “save” is pressed the path for the image is stored in the imagePath String variable.

addImage.addActionListener(

new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser file = new JFileChooser();

File workingDirectory = new File(".");

file.setCurrentDirectory(workingDirectory);

FileNameExtensionFilter filter = new FileNameExtensionFilter("\*.Images", "jpg","gif","png");

file.addChoosableFileFilter(filter);

int result = file.showSaveDialog(null);

if(result == JFileChooser.*APPROVE\_OPTION*){

File selectedFile = file.getSelectedFile();

book.setImagePath(selectedFile.getAbsolutePath());

}

}

}

);

For displaying the selected image a JButton named seeImage was created.



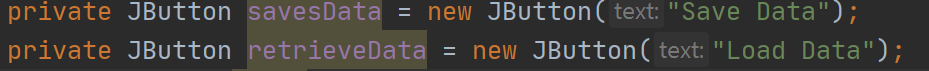
When it is pressed, a modal JDialog pops up with the title set to be the title of the book and with ImageIcon displayed in a JLabel. Image is obtained from the address stored in a imagePath String variable.



**Data Management**

This feature allows the user to store and load the book records from a so-called database. In this program, text file data.txt plays a role of a database. The feature is available on the main fraim. The feature consists of 2 parts: storing the data and retrieving the data from a text file.

Two buttons are created for this feature:



For those buttons, an inner class Action Listener was created. When the savesData button is pressed, it firstly checks whether the booklist (MyLinkedList) is empty and if so it aborts the operation and shows an error message.If booklist contains books, the program will write information about those books in a text file according to pre-defined order. For writing purpose, BufferedWriter and its method write() is used.

Try-catch is used to handle the unexpected IOExceptions.

The outer for loop traverses through every book object in the booklist and writes information about each book according to defined order. Later this order will be used for data retrieval. In case a particular field is empty for a book (imagePath or waiting queue), new lines are written in a text file. Otherwise, the information about the image path for a book is also written. When it comes to the waiting queue, if it is non-empty for a book, the inner for loop is used to traverse through and write each record. Records from the waiting queue are separated with semicolons. In “finally” statement, the BufferedReader is closed properly.

else if(e.getSource() == savesData){

if (bookList.isEmpty()) {

JOptionPane.*showMessageDialog*(Frame.this, "Error: Database is empty");

return;

}

BufferedWriter writer = null;

try{

writer= new BufferedWriter(new FileWriter(dataPath));

for(Book entry: bookList){

writer.write(entry.getISBN()+"\n");

writer.write(entry.getTitle()+"\n");

writer.write(entry.isAvailable()+"\n");

writer.write(entry.getImagePath()+"\n");

if(entry.getReservedQueue().getList().isEmpty())

{

writer.write("\n");

}

else{

for(String name:entry.getReservedQueue().getList())

{

writer.write(name+";");

}

writer.write("\n");//extra

}

}

}catch (IOException ex){

ex.printStackTrace();

}

finally {

if(writer!=null)

{

try {

writer.close();

} catch (IOException exception) {

exception.printStackTrace();

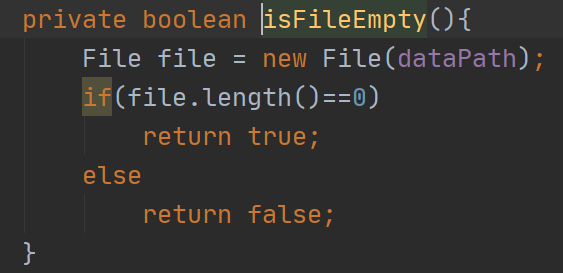
}

}

}

}

The next time an application is runned, the user has an option to load stored data from a text file. When the retrievesData button is pressed, it first checks whether the file is empty with the help of isFileEmpty() method to prevent error. This method checks and makes decisions about whether the file is empty or not based on the size of the file and returns boolean value accordingly.



If the file appears to be non-empty, BufferedReader is used to read the data from a text file line by line. The while loop is used to repeatedly read lines from a file and switch case is used to identify the order of the lines. Data is retrieved from a text file and stored in temporary String variables. Each 5 lines store information about a particular book and when the counter reaches 5, the book object is created with information retrieved and the counter is reseted. During this process of book object creation, the program checks whether there is information about the waiting queue. If there is information about the waiting queue, the program creates an array by using split method and “;” as regex. Then, a temporary object of MyQueue that stores strings is created. For loop is used to “enqueue” strings from an array into a newly created waiting queue. Finally, the waiting queue is assigned to a book object and a book object is added to the booklist.

Once the reader reaches the end of the file, the while loop breaks. Then, the table is updated according to information in a booklist.

The BufferedReader is surrounded with try-catch to handle IOException. Once the operation is over, the reader is closed properly.

else if(e.getSource()==retrieveData)

{

if(!isFileEmpty())

{

BufferedReader reader = null;

try{

reader = new BufferedReader(new FileReader(dataPath));

String line;

String tempisbn ="";

String temptitle="";

String tempavailable="";

String tempimagepath="";

String tempWaitQ = "";

int count = 0;

while(true){

if(count>4)

{

count=0;

Book temp = new Book();

temp.setISBN(tempisbn);

temp.setTitle(temptitle);

temp.setAvailable(Boolean.*parseBoolean*(tempavailable));

temp.setImagePath(tempimagepath);

if(tempWaitQ.length()!=0)

{

String[] arr = tempWaitQ.split(";");

MyQueue<String> q = new MyQueue<>();

for(String str:arr)

{

q.enqueue(str);

}

temp.setReservedQueue(q);

}

bookList.add(temp);

}

if((line=reader.readLine())==null)

break;

switch (count){

case 0:

tempisbn = line.trim();

break;

case 1:

temptitle = line.trim();

break;

case 2:

tempavailable = line.trim();

break;

case 3:

tempimagepath = line.trim();

break;

case 4:

tempWaitQ = line.trim();

break;

}

count++;

}

//System.out.println(bookList);

showAllRecords(bookList);

}

catch (IOException exception)

{

exception.printStackTrace();

}

finally {

if(reader!=null)

{

try {

reader.close();

} catch (IOException exception) {

exception.printStackTrace();

}

}

}

}

}

**Program Testing**

**Video:**

**<https://www.youtube.com/watch?v=TH-IXXA--jQ>**

**Conclusion**

To conclude, the second laboratory was extremely useful as it provides an opportunity to practice theory knowledge covered in lectures. While working on this laboratory I have again solidified my knowledge of object oriented design and programming. Moreover, I was able to solidify my understanding of basic data structures like linked lists and queues by implementing them from scratch. For this lab I was actively using my knowledge in Java Swing and event-driven programming concepts. In addition, I spent a significant amount of time studying some Java GUI concepts required for the program by reading official oracle documentations and analyzing sample codes.

I was able to design the application with UML diagrams and then proceed to its implementation and testing phases. Although I was working alone on the project and this means a lot of workload for me alone, I was able to get the maximum experience from the lab. In addition, I was using git version control in order to save intermediate results and simulate the process of working in a team. I perceive this laboratory to be a close simulation of working on a real project in an industry.

I believe my program meets the requirements since it works as it is supposed to. I also believe that I have met the expected outcomes of the lab.

**Future Development**

While working on this program I tried to directly follow the lab manual. Now I realize that program still has a lot of flaws and therefore in the future the following can be done:

1. User input could be validated in the way that ISBN should only accept the digits. This can be achieved by using the regex
2. The code can be further refactored for enhanced readability.
3. For storage purposes, mysql database can be used. In order to do this simple database design and some knowledge of jdbc are required
4. Images can be made flexible in size with Image class and getImage method.