**Library\_Admin\_System**

**Introduction**

Objectives:

* Programming Knowledge
  + Understand the principles of Object-Oriented design.
  + Apply Java in Object-Oriented software development.
  + Apply UML in Object-Oriented software modelling.
  + Apply an Object-Oriented approach to developing computer software.
  + Learn independently and be able to search for the information required in solving problems.
* Programming Practise
  + Use Eclipse IDE to write a Java program
  + Apply GUI and Data Structures with Java.
  + Use test cases to test the programming stability
  + Use block diagram to present ideas and findings effectively
  + Write a report to summarize the workflow and intensify programming knowledge
* Project Management
  + Divided task into different sub-task and plan scheduling
  + Work as a team and collaborate effectively with others.

Requirements of program:

Write a program to develop a Library Admin System.

Basic features:

* Initial UI
* MyLinkedList / MyQueue / Book class for data structures
* Add / Edit / Delete / Search / Load Test Data / Display / Sorting Book records stored in system
* Borrow/Return/Reserve books for library users

Additional features:

* Provide an image for each Book record and display in the page after clicking “More>>” button, setting the image by providing user option box
* Save and Retrieve the Book records as well as the Book image(if any) to/from file or database
* The JTable can select row by Keyboard (Up and Down Key)
* The search result also can be sorted

It can be achieved by a typical Java GUI application and generic classes through design, implementation, and testing phases.

Brief description of the methodology:

|  |  |
| --- | --- |
| Class | Class description |
| Frame\_mainGUI | A subclass of JFrame that provide GUI using frames, panels,  and simple GUI components. It also contains its own variables and functions to allow the user to control the system. |
| Frame\_moreGUI | A subclass of JFrame that provide |
| Book | A class contains its own variable that describes the book information |
| MyList<E> | A generic interface extended from Collection contains some abstract methods |
| MyLinkedList<E> | A generic subclass that contains its own variable, inner classes, function and implements the functions declared in MyList<E>. It is used to maintain a linked list of Book |
| MyQueue<E> | A generic class that contains a MyLinkedList<E> object and its own functions. It is used for the queue of waiting list |
| TestBook | A class provides the test case for Book and MyLinkedList<E> |
| TestMyLinkedList | A class provides the test case for MyLinkedList<E> |
| TestMyLinkedListExtra | A class provides the test case for MyLinkedList<E> and MyList<E> |
| TestMyQueue | A class provides the test case for MyQueue<E> |

**Methodology**

Division of work

The works are into two main tasks, i) program construction and ii) program testing

i. Program construction

* Program Flow
  + Construction of interface MyList<E> and its subclass MyLinkedList<E> for the data structure
  + Implement the button “Add”, ”Edit”, ”Delete”, ”Search”, ”Load Test Data”, ”Display All”, ”Display All By ISBN”, “Display All By Title”, “Exit” for data and system control
* UI Main
  + Construction of GUI of class Frame\_mainGUI and Frame\_moreGUI
  + Construction of subclass MyQueue<E>
  + Implement the “more” option that can borrow, return and reserve books for library users
  + Implement the additional features(Image Retrieval, Data Management)

ii. Program testing

* Program Flow
  + The correctness of the function implemented in “MyList”, “MyLinkedList”, “MyQueue”,”Book” and “Frame\_mainGUI”
    - Equivalence Partitioning Test for different cases (Testing with “TestMyList.java”, “TestMyLinkedList.java”, “TestMyQueue.java”,”TestBook.java” and “Load Test Data” in Frame\_mainGUI)
  + User Input checking for the GUI display
    - Negative Test to check if the function can show the error when the user inputs the wrong value or perform unexpected operations (eg. enter blank ISBN, operate “Edit”,”Delete” when database is empty)
    - Positive Test that the JTable shown expected result when the input correct information or perform normal operations
* UI Main
  + The correctness of the methods implemented in “Frame\_mainGUI”, “Frame\_moreGUI”, “Book”, and additional features “saving and loading data from database file”, “retrieval book image from existing file”.
    - Equivalence Partitioning Test for different operation(Testing buttons, text area and panels updates, sub-frame for “More” function with “ButtonListener”, “WindowListener” in Frame\_mainGUI)
    - User Input checking for the GUI display
      * Negative Test to check if the function can show the error when the user inputs the wrong value or perform unexpected operations (eg. load broken database with unstructured format, select improper image format)
      * Positive Test that the FrameGUI shown expected result when the user perform additional features operation(e.g. read data from database file and display correctly in the table, user reserved book in MoreGUI and will be updated to FrameGUI when MoreGUI is closed, image will be automatically loaded to MoreGUI if image path is existed)

The schedule of implementing the program

|  |  |
| --- | --- |
| Task | Time estimation |
| 1. Study the problem and instructions | 30 minutes |
| 1. Work on the individual part | 5 hours |
| 1. Integrate the parts | 2 hours |
| 1. Test and debugging | 30 minutes |

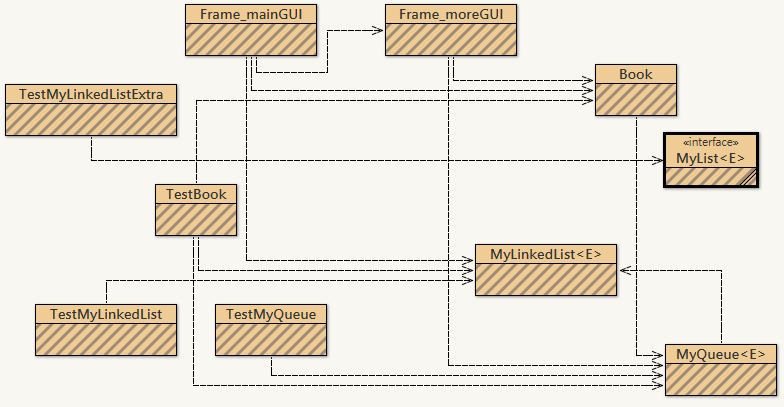
The program structure

The specifications of the classes defined, and the public/private member functions/variables included:

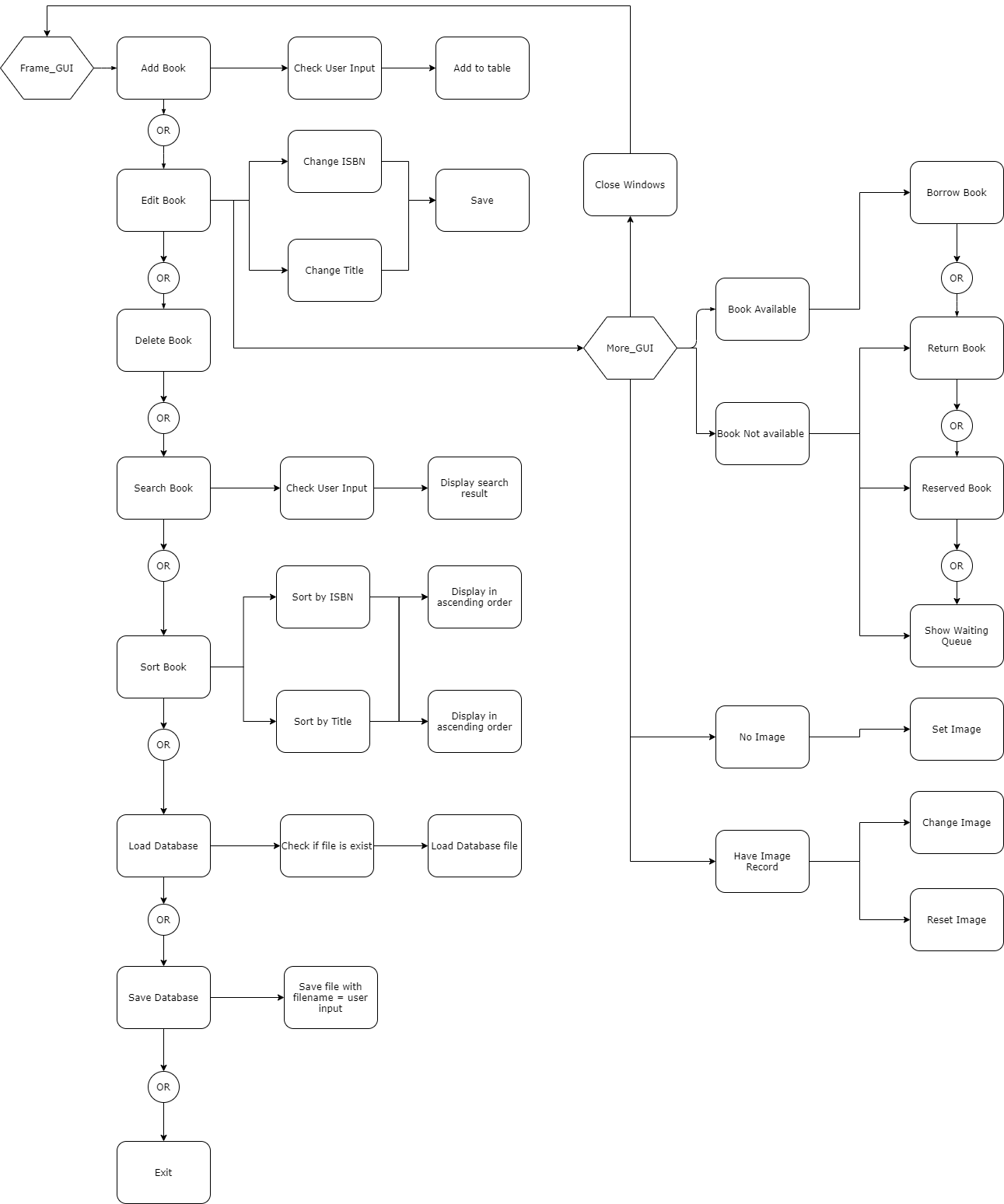
|  |  |
| --- | --- |
| Class | Class structure description |
| Frame\_mainGUI | public class Frame\_mainGUI extends JFrame: Subclass of JFrame. It inherits the methods of JFrame. It contains main() to provide user interfaces using frames, panels and GUI components. The JButton objects are added ButtonListener which are implemented to perform the features of Library Admin System. The JTable would update when the records changed / sorted.   * Data fields:   + private MyLinkedList<Book> bookLinkedList;   + private Book[] bookArray;   + private boolean reverseTitle;   + private boolean reverseISBN;   + private Vector<Vector<String>> rowData;   + private Vector<String> columnNames;   + DefaultTableModel model;   + private int status;   + private Book preChangeBook;   + private boolean searchMode   + private JFrame jf\_FrameGUI;   + private JTextArea jta\_AboutMe;   + private JScrollPane jsp\_ScrollList;   + private JTable jtb\_BookTable;   + private JPanel jp\_Control;   + private Frame\_moreGUI jf\_MoreGUI;   + private JLabel jlb\_ISBN;   + private JTextField jtf\_ISBN;   + private JLabel jlb\_Title;   + private JTextField jtf\_Title;   + private JButton jbt\_Add;   + private JButton jbt\_Edit;   + private JButton jbt\_Save;   + private JButton jbt\_Delete;   + private JButton jbt\_Search;   + private JButton jbt\_More;   + private JButton jbt\_LoadTestData;   + private JButton jbt\_DisplayAll;   + private JButton jbt\_DisplayAllByISBN;   + private JButton jbt\_DisplayAllByTitle;   + private JButton jbt\_Exit;   //Additional features   * + private JButton jbt\_LoadFromFile;   + private JButton jbt\_SaveToFile;   + private JTextField jtf\_FileName;   + private JButton[] buttonGroup;   + private String jointKey\_ISBN;   + private String jointKey\_Title;   + private String jointKey\_Available;   + private String jointKey\_Path; * Methods:   + public Frame\_mainGUI()   + public static void main(String[] args);   + private void init\_SubPanel\_AboutAuthor();   + private void init\_SubPanel\_BookListTable();   + private void init\_SubPanel\_ControlButton();   + public void init\_Button\_Listener();   + public void timeUpdate();   + public void updateTable();   Inner class:  class ButtonListener implements ActionListener   * Methods:   + public void actionPerformed(ActionEvent e)   + public void fn\_add()   + public void fn\_Edit()   + public void fn\_Save()   + public void fn\_Delete()   + public void fn\_Search()   + public void fn\_More()   + public void fn\_LoadTestData()   + public void fn\_DisplayAll()   + public void fn\_DisplayAllByISBN()   + public void fn\_DisplayAllByTitle()   + public void fn\_LoadFromFile()   + public void fn\_SaveToFile()   //for adding data from database to table   * + public void fn\_Add(boolean status, String imgpath, String[] queue) |
| Frame\_moreGUI | public class Frame\_moreGUI extends JFrame: Subclass of JFrame. It inherits the methods of JFrame. It contains a constructor Frame\_moreGUI() to provide user interfaces using frames, panels and GUI components. The JButton objects are added ButtonListener which are implemented to perform the features for editing the book availability, waiting queue and the image retrieval. The edited information will be saved to BookLinkedList when closing the JFrame, by implementing the “WindowListener”   * Data fields:   + private JFrame jf\_FrameGUI;   + private JTextArea jta\_AboutBook;   + private JPanel jp\_Control;   + private JTextArea jta\_SystemMessage;   + private JButton jbt\_Borrow;   + private JButton jbt\_Return;   + private JButton jbt\_Reserve;   + private JButton jbt\_WaitingQueue;   + private JPanel jp\_Image;   + private JButton jbt\_SetImage;   + private JButton jbt\_ResetImage;   + private Book MyBook;   + private MyQueue<String> BookQueueList; * Methods:   + public Frame\_moreGUI(Book inBook)   + public boolean getBookStatus()   + public MyQueue<String> getBookQueue()   + public String getBookImagePath()   + void init\_SubPanel\_AboutBook()   + void init\_SubPanel\_ControlList()   + void init\_SubPanel\_SystemMessage()   + void init\_SubPanel\_Image()   + void enableBorrow()   + void disableBorrow()   + void update\_jtextAboutBook()   + void update\_jtextSystemMessage(String strText)   + public void init\_Button\_Listener()   Inner class:  class ButtonListener implements ActionListener   * Methods:   + void fn\_Borrow()   + void fn\_Return()   + void fn\_Reserve()   + void fn\_WaitingQueue()   + void fn\_SetImage()   + void fn\_ResetImage()   + String getFileExtension(String file) |
| Book | public class Book: A class has its own variables to store ISBN, title and the status of Book with getter and setter function.   * Data fields:   + private String title;   + private String ISBN;   + private boolean available;   + private String path; * Methods:   + public Book()   + public Book(String ISBN,String title)   + public String getTitle()   + public String getISBN()   + public boolean isAvailable()   + public MyQueue<String> getReservedQueue()   + public String getPath()   + public void setTitle(String title)   + public void setISBN(String ISBN)   + public void setAvailable(boolean available)   + public void setPath(String path) |
| MyList<E> | public interface MyList<E> extends java.util.Collection<E>: An generic interface extended from Collection<E> that requires a class which stores elements in a sequential order to use this interface to implement its abstract function.   * Methods:   + public void add(int index, E e);   + public E get(int index);   + public int indexOf(Object e);   + public int lastIndexOf(E e);   + public E remove(int index);   + public E set(int index, E e);   + public default boolean add(E e)   + public default boolean isEmpty()   + public default boolean remove(Object e)   + public default boolean containsAll(Collection<?> c)   + public default boolean addAll(Collection<? extends E> c)   + public default boolean removeAll(Collection<?> c)   + public default boolean retainAll(Collection<?> c)   + public default Object[] toArray()   + public default <T> T[] toArray(T[] a) |
| MyLinkedList<E> | public class MyLinkedList<E> implements MyList<E>: Generic subclass of MyList<E>.It implements abstract methods of MyList<E> and has its own inner class Node and function to store the elements. It also has its own iterator to traverse all the elements in the list.   * Data fields:   + protected Node<E> head, tail;   + protected int size = 0; * Methods:   + public MyLinkedList()   + public MyLinkedList(E[] objects)   + public E getFirst()   + public E getLast()   + public void addFirst(E e)   + public void addLast(E e)   + public void add(int index, E e)   + public E removeFirst()   + public E removeLast()   + public E remove(int index)   + public String toString()   + public void clear()   + public boolean contains(Object e)   + public E get(int index)   + public int indexOf(Object e)   + public int lastIndexOf(E e)   + public E set(int index, E e)   + public int size()   Inner class:  protected static class Node<E>   * Data fields:   + protected E element;   + protected Node<E> next; * Methods:   + public Node(E o) |
| MyQueue<E> | public class MyQueue<E>: A generic class composite with MyLinkedList<E>. It has a MyLinkedList<E> object and uses its own function to perform operation of the list to achieve First In First Out (FIFO) data structure to represent the waiting list.   * Data fields:   + private MyLinkedList<E> list; * Methods:   + public void enqueue(E e)   + public E dequeue()   + public int getSize()   + public MyLinkedList<E> getList()   + public String toString() |
| TestBook | A class with main() to control workflow. It creates a MyLinkedList<Book> object and some Book objects to display the testing result.   * Methods:   + public static void main(String[] args); |
| TestMyLinkedList | A class with main() to control workflow. It creates a MyLinkedList<String> object and displays the result.   * Methods:   + public static void main(String[] args); |
| TestMyLinkedListExtra | A class with main() to control workflow. It creates a MyLinkedList<> object and upcast to MyList<String> to display the result.   * Methods:   + public static void main(String[] args); |
| TestMyQueue | A class with main() to control workflow. It creates a MyQueue<> object and displays the result.   * Methods:   + public static void main(String[] args); |

**The flow of execution**

Class diagram



Flowchart



**Program Testing**

|  |  |  |
| --- | --- | --- |
| Class | Test Case | Status |
| TestMyLinkedList | Use the given TestMyLinkedList to test the implementation of MyLinkedList class.  The result should be the same as the sample output if the implementation is correct.  The sample output:  (1) [America]  (2) [Canada, America]  (3) [Canada, America, Russia]  (4) [Canada, America, Russia, France]  (5) [Canada, America, Germany, Russia, France]  (6) [Canada, America, Germany, Russia, France, Norway]  (7) [Poland, Canada, America, Germany, Russia, France, Norway]  (8) [Canada, America, Germany, Russia, France, Norway]  (9) [Canada, America, Russia, France, Norway]  (10) [Canada, America, Russia, France]  (11) CANADA AMERICA RUSSIA FRANCE  (12) iterator.next(): Canada  (13) iterator.next(): America  (14) iterator.next(): Russia  (15) list after iterator.remove(): [Canada, America, France]  Before clearing the list, the list size is 3  After clearing the list, the list size is 0 | Pass |
| TestMyLinkedListExtra | Use the given TestMyLinkedListExtra to test the implementation of MyLinkedList class.  -Input “Jane” for the name  -Input “0” for index.  The result should be the same as the sample output if the implementation is correct.  The sample output:  [Tom, Susan, Kim, George, Peter, Jean, George, Jane, Denise, Jenny, Susan, Kathy, Jane]  Enter a name: Jane  Enter an index: 0  Jane is in the list? true  name at index 0 is Tom  Jane is at index 7  Jane is at last index 12 | Pass |
| TestMyQueue | Use the given TestMyQueue to test the implementation of MyQueue class.  The result should be the same as the sample output if the implementation is correct:  The sample output:  (1) Queue: [Tom]  (2) Queue: [Tom, Susan]  (3) Queue: [Tom, Susan, Kim, Michael]  (4) Tom  (5) Susan  (6) Queue: [Kim, Michael]  The size of the queue is 2 | Pass |
| TestBook | Use the given TestBook to test the implementation of Book class.  The result should be the same as the sample output if the implementation is correct:  The sample output:  11111111 Book1: available  22222222 Book2: available  33333333 Book3: Not available  Reserved queue: Borrower1 Borrower2 Borrower3  44444444 Book4: available | Pass |
| Frame\_mainGUI | The correctness of programming flow and correct book records shown.   * Input “0001” for the ISBN and “Java Book” for Title, then Click ”Add”   =>JTable shows “0001”,“Java Book”,”true” at the top row. The text fields are clear.   * Click “Load Test Data”   => JTable add 3 rows of new Book records   * Enter ISBN Click “Edit”   => Disable all button except “Save” if ISBN is exist in record   * Click the book record in the JTable then Click “Edit”   => Disable all button except “Save” if ISBN is exist in record   * After enter edit mode, input new ISBN (the ISBN should different with other Book) and Title then Click “Save”   => Update JTable with modified information and clear the text fields.  => Disable the “Save” button and enable all the other buttons.   * Enter ISBN then Click “Delete”   => Remove the book with input ISBN from JTable and clear the text fields.   * Click the book record in the JTable then Click “Delete”   => Remove the selected book from JTable and clear the text fields.   * Input the keywords to be searched of ISBN or Title then Click “Search”   => Display the Book record that contains the keywords, change the status to “Search Result”   * Click “Display All”   => Display all book records following the order of adding Book to system, change the status to “Displaying All”.   * Click “Display All Books by ISBN”   => Display all book records in ascending order of Book ISBN, change the status to “Sorting”  => If previous status is ”Search Result”, display search result book records in ascending order of Book ISBN, change the status to “Sorting the result”.   * Click “Display All Books by ISBN” again   => Display all book records in reversed order of Book ISBN.  => If previous status is ”Sorting the result”, display search result book records in reversed order of Book ISBN.   * Click “Display All Books by Title”   => Display all book records in ascending order of Book Title, change the status to “Sorting”  => If previous status is ”Search Result”, display search result book records in ascending order of Book Title, change the status to “Sorting the result”.   * Click “Display All Books by Title” again   => Display all book records in reversed order of Book Title.  => If previous status is ”Sorting the result”, display search result book records in reversed order of Book Title.   * Enter ISBN then Click “More>>”   => Display a new modal dialog page of Frame\_moreGUI.  => Disable all button in Frame\_mainGUI   * Click the book record in the JTable then Click “More>>”   => Display a new modal dialog page of Frame\_moreGUI.  => Disable all button in Frame\_mainGUI   * Close dialog page of Frame\_moreGUI   => Refresh the JTable records with updated available status   * Click “Save To File”   => Display a message “Database is saved to: “C:\xxxxx\xxxxx\xxxxx\xxx.txt”   * Click “Load From File”   => Restart the program and click “Load From File”  => Display all data in the table  => Click “More” to see the book information  => Queue and Image can be shown correctly   * Click “Exit”   => Terminate the program without error | Pass |
| Negative Test to check if the program can catch the error.   * If the ISBN is empty and click ”Add” /”Edit”/ ”Delete” / “Save”   => Display the error message "Error: ISBN cannot be blank."   * If the ISBN not empty but Title is empty and click ”Add”/”Save”   => Display the error message "Error: ISBN cannot be blank."   * If the ISBN and Title are not empty but ISBN is not 10-digit number and click ”Add”/”Save”   => Display the error message "Error: ISBN should be a 10-digit number."   * If the ISBN and Title are not empty but the database contain the input ISBN when click ”Add” / “Save”   => Display the error message "Error: book ISBN(the input) is already in the database."   * If database is empty and click ”Edit”/“Delete”   => Display the error message "Error: Database is empty."   * If database does not contain the input ISBN when click ”Edit”/“Delete”/”More>>”   => Display the error message "Error: book ISBN (the input) is not in the database."   * If the ISBN is empty and click ”More>>”   => Display the error message "Error: Please select a book or input ISBN"   * If the Filename is empty and click ”Load From File” / ”Save To File”   => Display the error message "Error: File Name cannot be blank." | Pass |
| Exceptional Test to check if the program can catch the exception.   * If the ISBN and Title are not empty but ISBN is non-number and click ”Add”   => Display the error message "Error: ISBN is not a number."   * If the file does not found as the input of Filename when click ”Load From File”   => Display the error message "Error: File not found."   * If the file is found but the format of content is incorrect when click ”Load From File”   => Display the error message "Error: Database is damaged."   * If no records in database when click ”Save To File”   => Display the error message "Error: No data recorded."   * If IO problem appear when click ”Save To File”   => Display the error message "Error: IO Exception." | Pass |
| Frame\_moreGUI | The correctness of programming flow and correct system message shown.   * If the Book object is available   =>The “Borrow” button is enabled, all the other basic buttons are disabled. (“Set Book Image” and “Reset Book Image” are independent from the basic buttons)   * If the Book object is unavailable   =>The “Borrow” button is disabled, all the other basic buttons are enabled.   * Click “Borrow”   => “Borrow” becomes disabled while all the other buttons become enabled.  => Display the updated available status at the top and display the system message "The book is borrowed." at the bottom.   * Click ”Reserve”   => Display an input dialog to prompt for the user’s name. The input will be added to the reservedQueue. => The system message "The book is reserved by (input) will be displayed at the bottom.  .   * Click ”Waiting queue”   => Display the reserved queue of the Book object at the bottom.   * Click ”Return” and no user in the reserved queue of the Book object   => The available status will be turned to true.  => The “Borrow” button becomes enabled and all the other buttons become disabled.   * Click ”Return” and some user/users in the reserved queue of the Book object   => Removed the head user from the queue. Display system message “The book is returned. The book is now borrowed by (the head user’s name)." at the bottom.   * Click ”Set Book Image”   => The JFileChooser dialog shown to prompt for the image path. If the extension equals ".jpg” or “.gif”, the image will be shown in the JFrame and store the path to Book record  => Display the system message “Image from (imagePath) is set.” at the bottom.  => The button name will change to “Change Book Image” and “Reset Book Image” become enabled   * Click ”Reset Book Image”   => Clear the image shown in the JFrame  => Clear the path stored in Book record  => “Reset Book Image” become disabled and the button ”Change Book Image” will reset name “Set Book Image” | Pass |
| Exception Test to check if the program can catch the exception.   * Select the image/files with the extension not equals ".jpg” or “.gif” when JFileChooser dialog shown   => Display the error message "Error: Wrong image format." | Pass |

**Program Testing**

Video Link:

Part 1:

<https://youtu.be/qZwJUHh3jSY>

Part 2:

<https://youtu.be/ufJD1xetjg0>

**Conclusion**

The project provides our team practice to deepen our Java programming knowledge, such as using GUI components to develop user interfaces for Java applications, and also using generic class. We also learn how to manage the program progress as a team project. Besides, it is also a good opportunity for us to get familiar with the Eclipse environment.

After finishing the project, we have experienced the progress for designing an object-oriented Java program as a beginner. From problem analysis to programming design, followed by task distribution and test case design, then go to programming and debugging, finally for the report documentation and demonstration. The most difficult part we faced is to design the possible fail/exception test case and any future development that could be made to our program, as there will be lots of improper combinations that could crash the program. As there are some additional functions added to the program which do not have any rules or guidelines to follow, we browse the internet and discuss together to find out the best solutions to perform in our program. Finally, we complete the additional functions and ,of course, it is not perfect and could have further improvements to enhance the program.

Overall, it really gives us a good experience of programming knowledge, as well as project management skills.

**Future Development**

The program can have further development on below a few points.

1. Improvement of data structure stored in database
   1. Currently
      1. the database file is stored as .txt file
      2. each single line in txt file contains one book’s information
      3. the book information is stored in format :   
         BookISBN + “joint\_key” + BookTitle + “joint\_key” + BookIsAvailable + “joint\_key” + BookImagePath + “joint\_key” + BookQueueList
      4. the book is set to be readable in program, however user may change the file to writable by right-click>property
   2. To increase data privacy,
      1. use of json or xml file format to store the database
      2. data are stored according to the <key><value> structure
      3. encrypt the database file saved by using private encoder, so that user cannot read the database file unless know the
      4. set file to hidden to avoid user edit or delete
2. Develop a local folder to store the Book image
   1. Currently,
      1. the image of Book are now stored in the different directory path of the PC
      2. there is a risk that the images may moved / deleted by others
      3. so that the Book image may lost
   2. To prevent this situation,
      1. we can create a local folder to store the image
      2. by copy the image to the local folder and
      3. set the directory to the local folder
3. Improvement of the security
   1. Currently,
      1. the system can be used directly without login
      2. anyone using the PC can control the system and edit the database
      3. it is unsafe if the exist of hacker
   2. To improve the security of system,
      1. we can set up a login function
      2. only the library admin know the account and login the system
      3. so that the system would not affected by other PC user