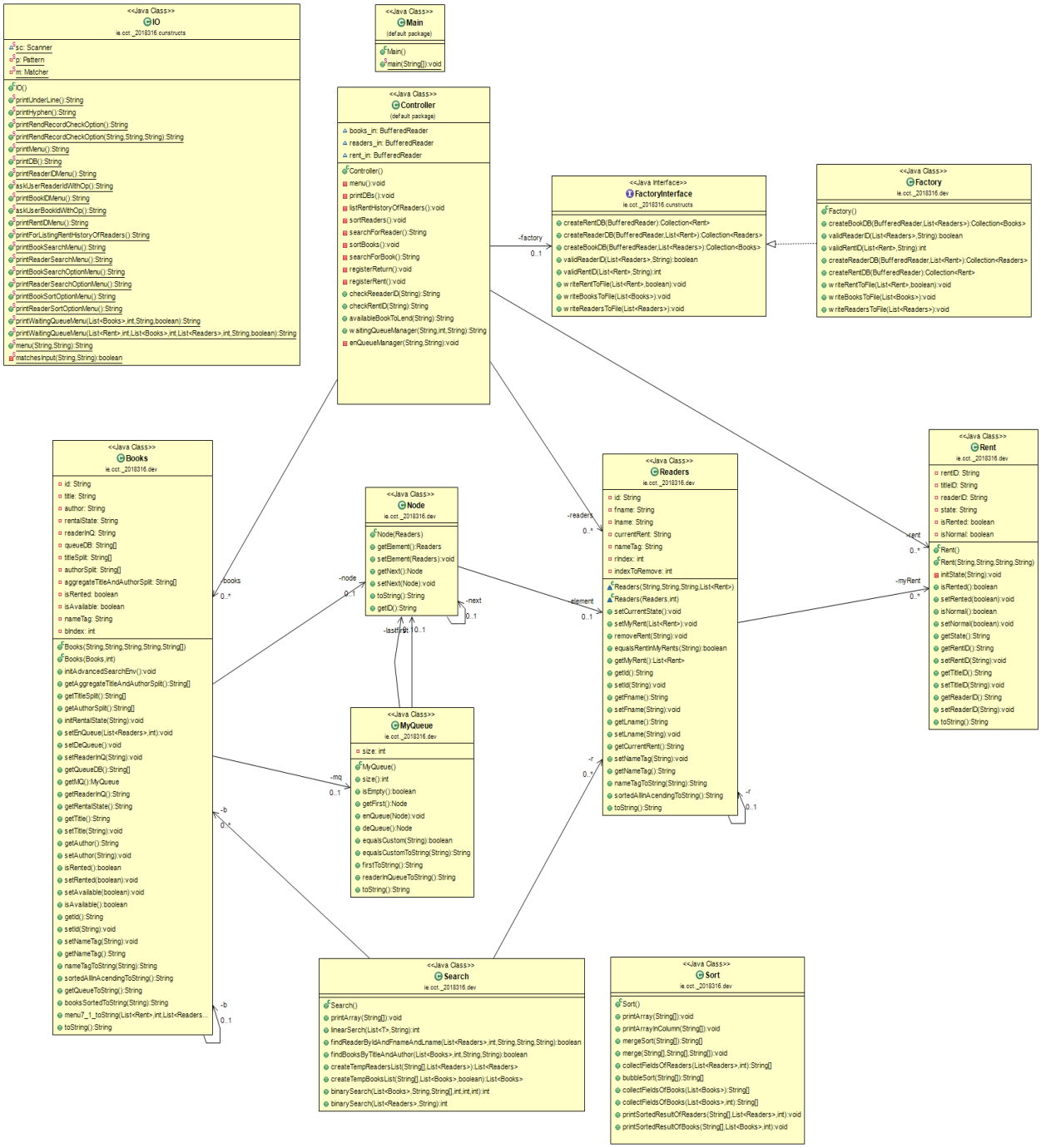
Short justification for logic design decisions of Library System.

The ObjectAid UML Explorer that is supported as Eclipse plugin is used to generate a class diagram for this project.

- Zoom in to have a close look at the class structures in detail.



Explanation of the structure of your code

Class Main

A main method is in it and the main method invokes the constructor of Controller

Class Controller

It is the class that manages controls the program in overall

The main menu consists of as follows

```
[Please select menu]
1: Search for books
2: List books
3: Search for readers
4: List readers
5: Register a rent
6: Register a return
7: List the books that readers have borrowed
8: Print DBs
9: Close program
```

The menu “8” can be used to check if txt files are loaded into the system on running the program.

Interface FactoryInterface

This interface has method signatures that are used for loading files and writing to files

Class Factory

The class Factory implements FactoryInterface . An instance of the class Factory is created on invoking the class Controller (as soon as the program starts to run)

Final class IO

It manages all the print-related input & output interaction between the system (the program) and user.

Class Books

- Fields

Main Fields shown at UI level	Fields that works behind
String ID (bookID) **String** Title **String** Author **String** Rental_state **String** readerInQ	**MyQueue** mq *//a book’s queue* **Node** node *//node(reader) in queue*

<An example of a book record shown at UI>

B01[id=B01, title=Hunger Roof, author=Georgi Facello, rental_state=Available, readerInQ=none]

B03[id=B03, title=Outbreak, author=Parto Bamford, rental_state=Rented, readerInQ=R03 R04 R05]

● Note

- All the book ID has a prefix of "**B**" before numeric value

- The field '**rental_state**' has two states.
  A book record itself does not consider who/which reader is renting the book.
  In other words, a book only store if available or being rented

Available	Rented
The book has never been rented or has been returned	The book is being rented at the moment

- The field 'readerInQ' has two states.

none	**R03 R04 R05**….
The book has no queue of readers	The book has queue of readers. In the example above, - 1st queue is R03 (readerID) - 2nd queue is R04 (readerID) - 3rd queue is R05 (readerID)

- All above is a brief explanation of the Class Books.
  There are some other fields which work behind and are not mentioned for now.
  Any detailed explanation can be found on the top of class or its method signatures.

Class MyQueue

This class is the custom queue implementation is in a **linked-list** approach.
Any detailed explanation can be found on the top of class or its method signatures in.

Class Node

This class works under the class MyQueue and holds nodes(reader OBJ)
Any detailed explanation can be found on the top of class or its method signatures.

Class Readers

● Fields

Main Fields shown at UI level	Fields that works behind
String ID (readerID) **String** first name **String** last name **String** current Rent	**List<Rent>** myRent *//a reader's current rent*

<An example of a reader record shown at UI>

R01[id=R01, fname=Dulce, lname=Abril, Current Rent=RT02 RT03 RT04]

R02[id=R02, fname=Mara, lname=Hashimoto, Current Rent=none]

● Note

- All the reader ID has a prefix of "**R**" before numeric value
- The field '**Current Rent**' has two states.

none	**RT02 RT03 RT04 ....**
The reader has no any rent record	The reader has current rents. In the example above, - the reader has three rent records and each of rent records is represented as its own rentID

- All above is a brief explanation of the Class Readers.
  There are some other fields which work behind and are not mentioned for now.
  Any detailed explanation can be found on the top of class or its method signatures.

Class Rent

● Fields

Main Fields shown at UI level	Fields that works behind
String ID (rentID) **String** titleID(bookID) **String** readerID **String** state	**boolean** isRented **boolean** isNormal

<An example of a rent record shown at UI>

RT01[rentID=RT01, titleID=B01, readerID=R05, state=Normal]

RT02[rentID=RT02, titleID=B03, readerID=R01, state=Rented]

● Note

- All the rent ID has a prefix of "**RT**" before numeric value
- The field '**state**' has two states.

Normal	**Rented**
The rent record has a titleID and a readerID. The reader represented as 'readerID' has rented the book represented as 'titleID' before. The book has successfully been returned at the moment.	The rent record has a titleID and a readerID. The reader represented as 'readerID' is renting the book represented as 'titleID' at the moment.

- All above is a brief explanation of the Class Rent.
  Any detailed explanation can be found on the top of class or its methods signatures

Class Search

- This class is used when searching for a book or a reader.
- The custom search algorithms used are **binary search** and **linear search**.

binary search	**linear search**
It's executed allowing multiple string inputs on a line when user enters search keywords.	It's executed taking an accurate string input at a time.

- This is a brief explanation of the Class Search.
- Any detailed explanation can be found on the top of class or its method signatures

Class Sort

- This class is used when sorting books or readers.
- The custom search algorithms used are **bubble sort** and **merge sort**.
- This is a brief explanation of the Class Sort.
- Any detailed explanation can be found on the top of class or its method signatures

Explanation of the structure of the text files used to ensure data persistency

<An example of the format of text files>

Books.txt	Readers.txt	Rent.txt
bookID|title|author|rental state|readerInQ B01#Hunger Game#George#Rented#R01 R02 R03 B02#Frogmen Breaking#Bezalel Simmel#Available#none B03#Outbreak#Parto Bamford#Rented#R03 R04 R05 B04#Young Language#Chirstian Koblick#Rented#none B05#Stampede Disturbing#Kyoichi Maliniak#Rented#none B06#World Leathernecks#Anneke Preusig#Available#none B07#Goodfellas Salute#Tzvetan#Available#none ---------- **[NOTE]** • Fields title and author title and author are based on **one word unit** without any space or **two word units** with a space between them. No more than two word units is allowed for these fields so as to keep it simple. It's recommended to follow this convention if changing value or adding value. • Fields title and author, rental state The 1st letter should always be capitalized	readerID | first name | last name | currentRent R01#Dulce#Abril#RT02 RT03 RT04 R02#Mara#Hashimoto#none R03#Philip#Gent#none R04#Kathleen#Hanner#none R05#Nereida#Magwood#RT01 R06#Gaston#Brumm#none R07#Etta#Hurn#none ---------- **[NOTE]** • Fields title and author title and author are based on **one word unit** without any space. No more than one word unit is allowed for these fields so as to keep it simple. It's recommended to follow this convention if changing value or adding value. • Fields title and author, rental state The 1st letter should always be capitalized • Field current rent **RT02 RT03 RT04** == current rent IDs If there's no current rents, its value is **none,** which all is lowercased and which is the default state.	rentID | bookID | readerID | State RT01#B01#R05#Rented RT02#B03#R01#Rented RT03#B04#R01#Normal RT04#B05#R01#Rented ---------- **[NOTE]** • Field state **Rented** == a reader(readerID) is currently renting a book(book/titleID). The reader will holds the rentID while renting **Normal** == the reader has rented the book before, the book has been returned. Rent.txt will have rent records only when readers rent a book.

• Field rental state **Rented** == is being rented **Available** == is returned or never been rented(default) • Field readerInQ **R01 R02 R03** == readers(readerID) in a book's queue If there's no reader in queue, its value is **none,** which all is lowercased and which is the default state. If there's reader in queue, the field takes reader ID(s) with a space. This is the field that holds readers in queue, NOT any reader who is currently renting this book OBJ • More rows can be added or duplicated as far as bookID is unique which means an incremented number with a prefix **B** in front of that. if adding/duplicating rows to just place dummy DB, it's recommended to set the field rental state to be "Available" and the field readerInQ to be "none" As the default state. Again, no current reader who is renting a book is recorded into Books.txt or a book OBJ in the system. That will be performed in Rent.txt or Readers.txt		

- The 1st line (line 0) is each of the fields descriptions.
- From the 2nd line (line 1), each of the records are represented.
- **#** is a delimiter that distinguishes fields each.
- All the rent ID has a prefix of "**RT**" before numeric value
- All the reader ID has a prefix of "**R**" before numeric value
- All the book ID has a prefix of "**B**" before numeric value
- Any ID's numeric value that is in 1s unit and that is less than 10, a zero padding must be set.
  (**This is very important and some wrong result can be returned when searching for/listing/sorting books or readers, rent records**)
  e.g. an example of wrong formatted 1s unit within unique IDs
  ~~B1 B2 B3 B4 B5 B6 B7 B8 B9~~
  ~~R1 R2 R3 R4 R5 R6 R7 R8 R9~~
  ~~RT1 RT2 RT3 RT4 RT5 RT6 RT7 RT8 RT9~~

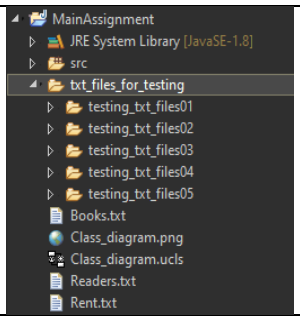
- It's STRICTLY not recommended to manually write to fields in files with special characters.
  Fields values in the most cases consist of alphabet letters except ID fields.
  Thus, Most of the regx are in the form of `[a-zA-Z0-9]++`

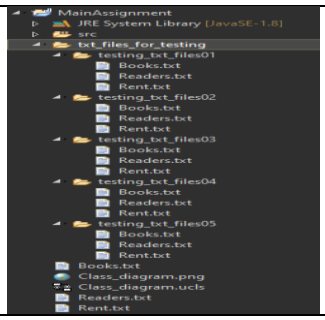
- It's STRICTLY not recommended to change any text files manually,
  since all changes are overwritten into each of the existing files when.
  (If really necessary, do so with a special care as far as you don't break the rule/syntax)

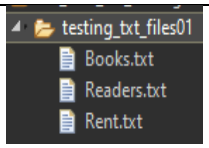
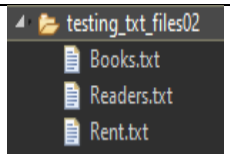
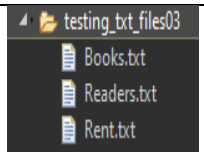
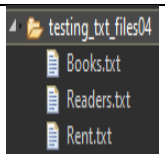
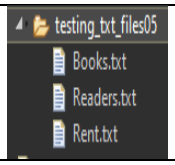
- Path of files should remain the same properly (The top-level path of the project folder).
  Otherwise, the program will throw **FileNotFoundException** and the program will crash.

- Each of the files is to be loaded into system as the program runs as far as file path is correct.

- Five sets of testing txt files in each of folder are attached in the folder named "**txt_files_for_testing**"

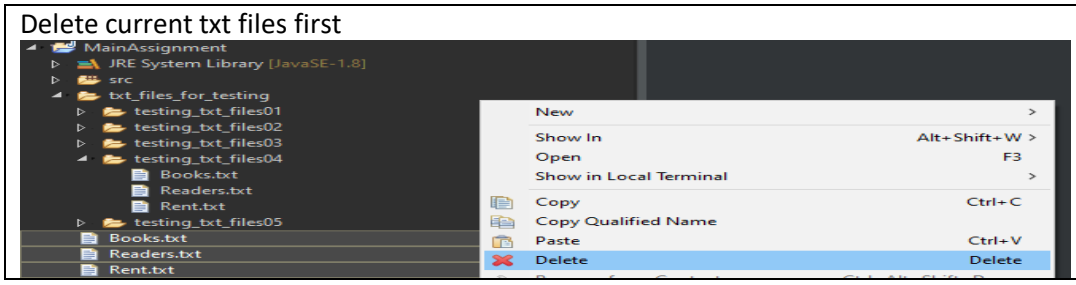




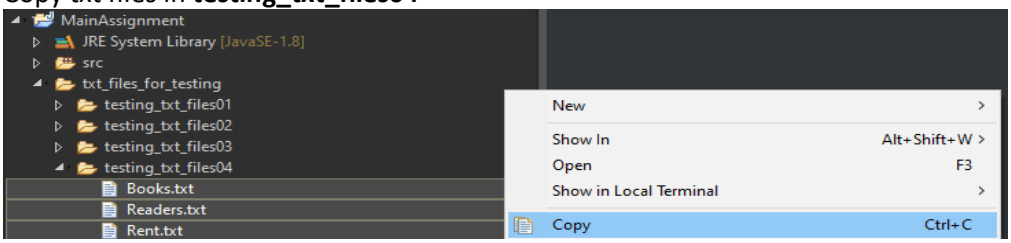
				
[testing_txt_files01] - a small file set - with a small testing fields values Assigned - This is set to be loaded into the system initially	**[testing_txt_files02]** - a small file set - with a small testing fields values assigned	**[testing_txt_files03]** - a small file set - with default states [Note] - Rent.txt has only one row to create a Rent OBJ and Rent list _____ - user can then run the program (play with the files loaded) and any change at the system level will be updated into the existing txt files.	**[testing_txt_files04]** - a file set (Books.txt == 100 rows) (Readers.txt == 100 rows) - with default states [Note] - Rent.txt has only one row to create a Rent OBJ and Rent list _____ - user can then run the program (play with the files loaded) and any change at the system level will be updated into the existing txt files.	**[testing_txt_files04]** - a file set (Books.txt == 1000 rows) (Readers.txt == 1000 rows) - with default states [Note] - Rent.txt has only one row to create a Rent OBJ and Rent list _____ - user can then run the program (play with the files loaded) and any change at the system level will be updated into the existing txt files.

- If using one of the testing txt file set,
  please be sure to put it into the proper location within the project folder.
  The safest and easiest way to use a txt file set is to copy and paste it into the project folder.
  For example, let's say we load a txt file set from the sub-folder **testing_txt_files04**.

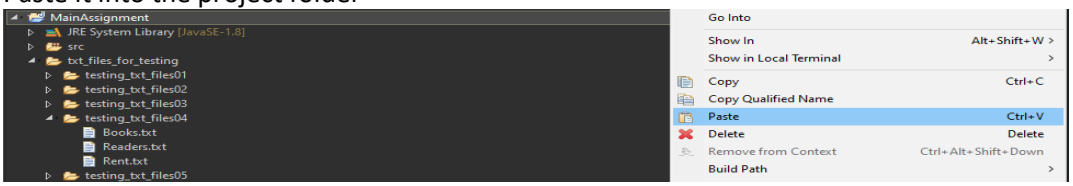
Delete current txt files first



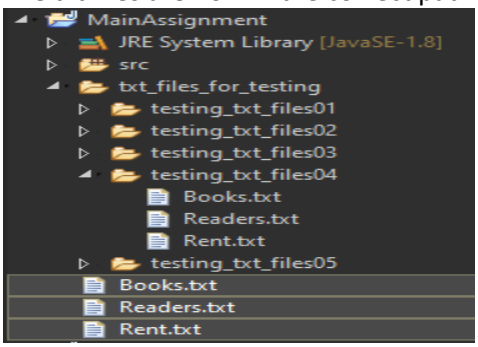
Copy txt files in **testing_txt_files04**

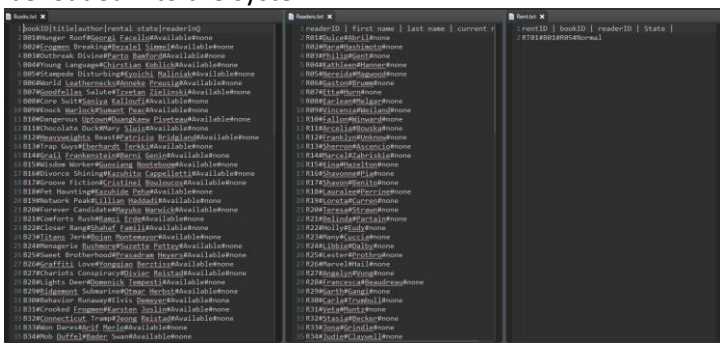


Paste it into the project folder



The txt files are now in the correct path to be loaded into the system





● Any other DB set can be used as far as following the syntax mentioned all above

Reference

Any code or inspired approach that is referenced is written on top of class or method signature.

Most of them are from the course contents that have been lectured by Professor Amilcar.

GIT

https://github.com/kyuing/LibrarySystem