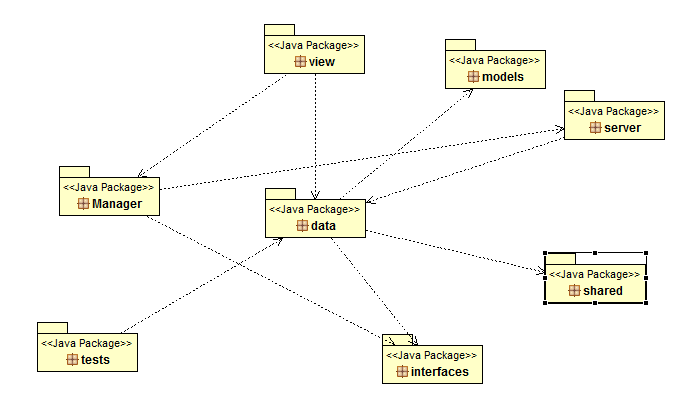
Design Documentation

Package Distributions:



<java package> data

This package contains all the methods manipulating the data like creating doctor or nurse and editing a record depending upon the record type, overall acting as a repository of data.

<java package> server

This package contains all the functionalities required by the server for the execution of distributed system through java RMI along with the execution of methods of interface definitions.

<java package> shared

This package contains the shared components distributed in the programming of the application like exception class specifying particular exception depending upon the condition or enumerations for nurse records consisting nurse designation and nurse status.

<java package> interfaces

This package contains interfaces with methods definition required for complete execution of distributed system.

<java package> tests

This package consists of all the tests required for the confirmation of functionalities of the code written for development of the application.

<java package> Manager

This package deals with the client side binding with the server over the distributed system and execution of server methods through interface definition.

<java package> view

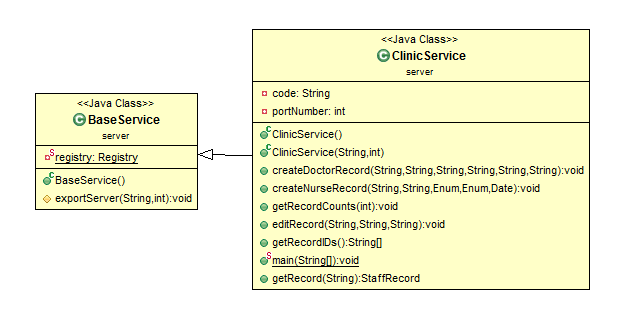
This package consists of all the classes that makes up the client side user interface of this application allowing the graphical execution of the operations for completion of this system.

<java package> models

This package contains all the model side information including the doctor record information or attributes or nurse record information or attributes.

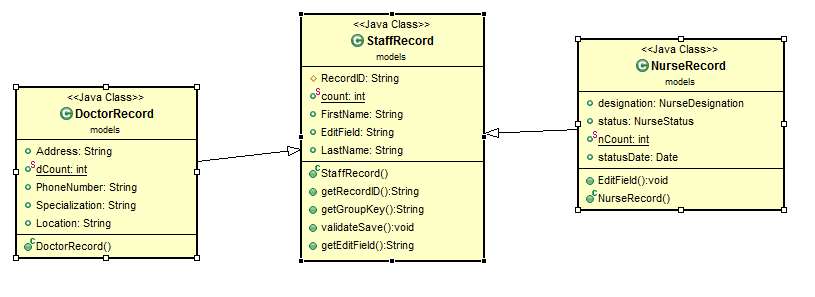
Implementation:

* Three servers associated with three different clinic locations.



Implemented through inheritance with BaseService.java as parent class containing the registry and exportServer method used to bind three servers with three different ports mentioned by the integer parameter. ClinicService is inherited class containing the functionality of BaseService.java for linking the servers with associated port numbers and implementing the client-server interface and functionality or execution of all the functions required in this assignment.

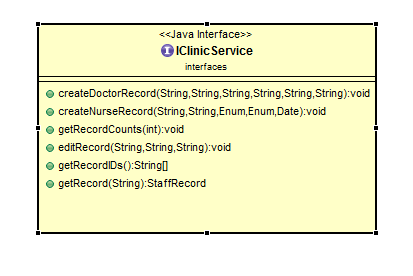
* Doctor Records and Nurse Records:

 Doctor records and Nurse records were executed along with the execution of inheritance. As both doctors and nurses made up staff, Staffrecord.java is the base class consisting the common attributes of both the sort of staff like FirstName, LastName and contains methods that were to be implemented on both sorts of records like getting the recordID,getting the group key on which hashmaps are to be grouped. For Doctor records , StaffRecords is inherited to class DoctorRecords.java with other attributes required for doctors and the constructor taking care of the unique recordID with “DR” prefix for doctor records. Similarly, Nurse records are made up by inheriting StaffRecord.java and other attributes mentioned in NurseRecord.java with constructor NurseRecord taking care of unique recordID with prefix “NR”.

* Client Side Execution:

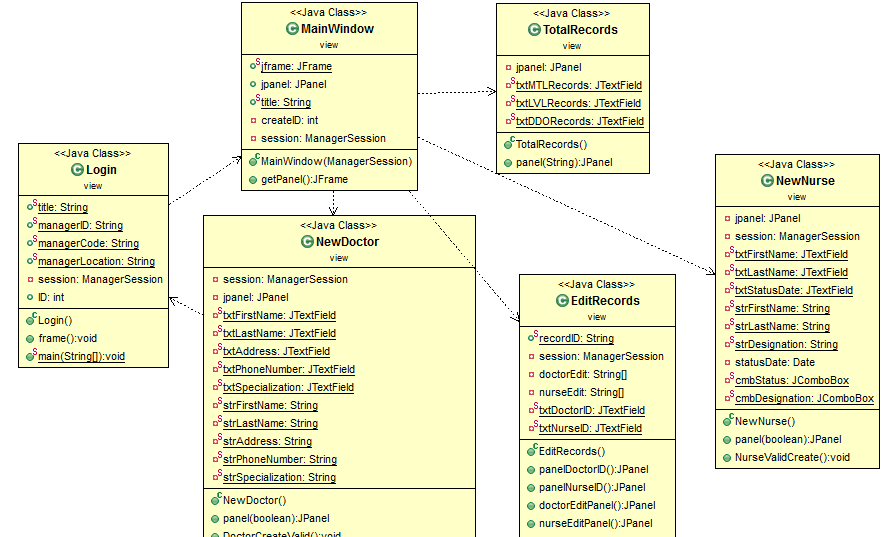
 ClientServiceFactory.java contains constructor used to set the rmi manager of the client and getService method is used to bind client to the server with implementation of Naming.lookup with particular servers and port numbers. ManagerClient.java class containing main method for the client used to divert the control to the user interface of the application. ManagerSession.java is used to create a session of the manager of particular manager with taking managerID as parameter and object reference to call to the server.

* Java RMI Interface:



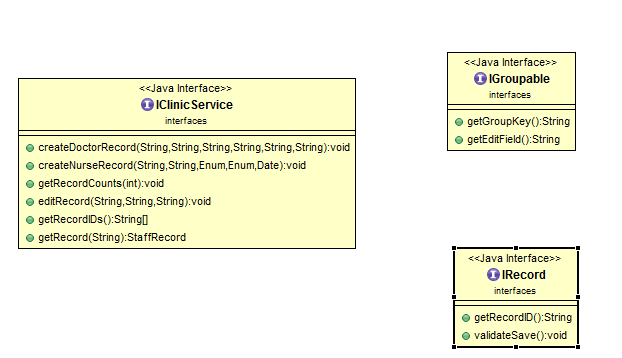
Java RMI interface definition for ClinicService (ClinicServer) containing the declarations of all the functions required to be implemented by the server.

* User Interface:



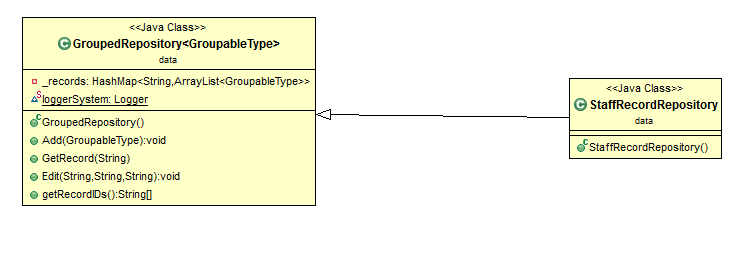
For user interface, the flow goes from Login.java , a class permitting the manager to input his id alongwith the clinic locations to access the operations allowed from this application to MainWindow.java class containing menu items for creating doctor, creating nurse, editing nurse, editing doctor and getting total number of records of doctors and nurse.NewDoctor.java containing JFrame specifications of attributes required for creating a doctor. Similarly NewNurse contains the same functionality for creating the nurse. EditRecords contains the functionality for editing both doctors and nurses.

* Interfaces



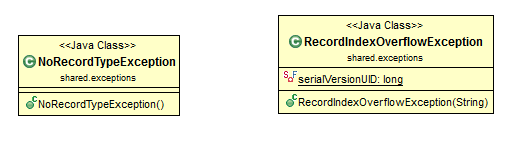
IClinicService is the interface definition required for the execution of operations by server invoked by client. IGroupable is another interface that will be applied to staffrecords and contains the method for grouping the key in hashmaps. IRecord is another interface that will be implemented by the repository so that staff members either doctor records or nurse records can extends it and implement operations for getting the recordID for further execution of operations.

* Data Repository



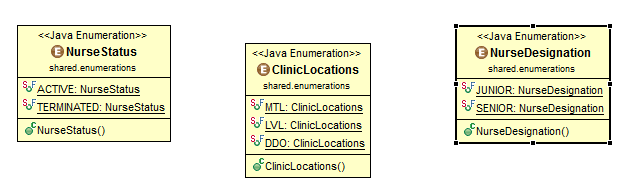
This package contains two java classes, GroupedRepository.java is the generic class extending IGroupable and IRecord as its wildcards. The main purpose of this class is to allow the execution of operations on both kinds of operations including doctor record and nurse record irrespective of type acting as generic operations execution class. StaffRecordRepository.java class extends GroupedRepository.java through which we can implement the operations of GroupedRepository by considering the objects of this class.

* Exceptions:



This section of application deals with the exceptions that might be arise from any other section of the application during their execution. Some of these exceptions are listed above like NoRecordTypeException exception that may be arose due to type other than doctor and nurse or RecordIndexOverflowException that will be arise when the records entered will take count more than 5-digit unique number.

* Enumerations:

This section of application deals with all the information that may be represented in enumerations having fixed values in their parameters. Some of the enumerations used in this application are listed above NurseStatus, ClinicLocations and NurseDesignation.

* **Data Structures & Techniques Used**:
* HashMaps

**private** HashMap<String, ArrayList<GroupableType>> \_records = **new** HashMap<String, ArrayList<GroupableType>>();

Hashmaps are used to group the records, Groupable here means any record extending from IGroupable and IRecord with String being first letter of last name. These records are stored in arraylists in form of instances of groupable. Basically, HashMap data structure is used to map the arraylist of instances with special string key.

* ArrayLists:

ArrayList<GroupableType> existingGroup = \_records.get(key);

existingGroup.add(record);

The arraylist, data structure is used to store the instances of Groupabletype, in this application GroupableType may be Doctor Record or Nurse Record.

* Generics:

Generics in this application is implemented to classes and functions to execute what is desired in this application.

**public** **class** GroupedRepository<GroupableType **extends** IGroupable & IRecord>

Groupedrepository is the generic class that will take any type GroupableType as its type that will be extended by IGroupable and IRecord.

**public** **void** Add(GroupableType record)

{

ArrayList<GroupableType> newGroup = **new** ArrayList<GroupableType>();

newGroup.add(record);

}

Methods to create nurse or doctor is a function taking generic type as its parameter and can be used to store the record again in arraylist of generic type.

**public** **void** Edit(String recordID, String FieldName, String FieldValue) **throws** Exception

{

GroupableType record = GetRecord(recordID);

Field[] recordFields = record.getClass().getFields();

**if**(field.getName().equals(FieldName))

{

**if**(field.getType().toString().equals(FieldValue))

{

field.set(record, FieldValue);

Editing method for doctor record or nurse record is also generic beginning with attaining the record and afterwards using java reflection to get the field of the object and then validating that field to the field to be edited and after validations assigning the new value to that field.

**Difficulties Found:**

* Concurrency:

Dealing with concurrency when multiple clients will be using the application and invoking the same function is the main focus to deal with in terms of concurrency. But the layering or the architecture of the application allows the operating system to take care of concurrency as depicted by our experiment.



Last two tabs in application shows management of concurrency.

Ctest1 : creates doctor records after a delay of 1 second each

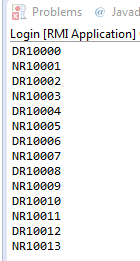
Ctest2 : creates 7 records of nurses after a delay of 1 second each

Running Ctest1 from one client and Ctest2 from another client gives the following results which is evident for concurrency management on the application.

After this ,



Record info shows the concurrency of records



* HashMaps:

Storing the records with multiple attributes and of different types and attaining is also one of the difficulties, we found on our way for completion of this application.

* Editing Records:

Attaining a particular record from the hashmaps and that too with following search was also an issue which was handled by java reflection. But learning and implementing java reflection for getting the records and validating the fields was also a crucial part of the difficulties found.