SE311 – Virus Tracking Application – Project Report

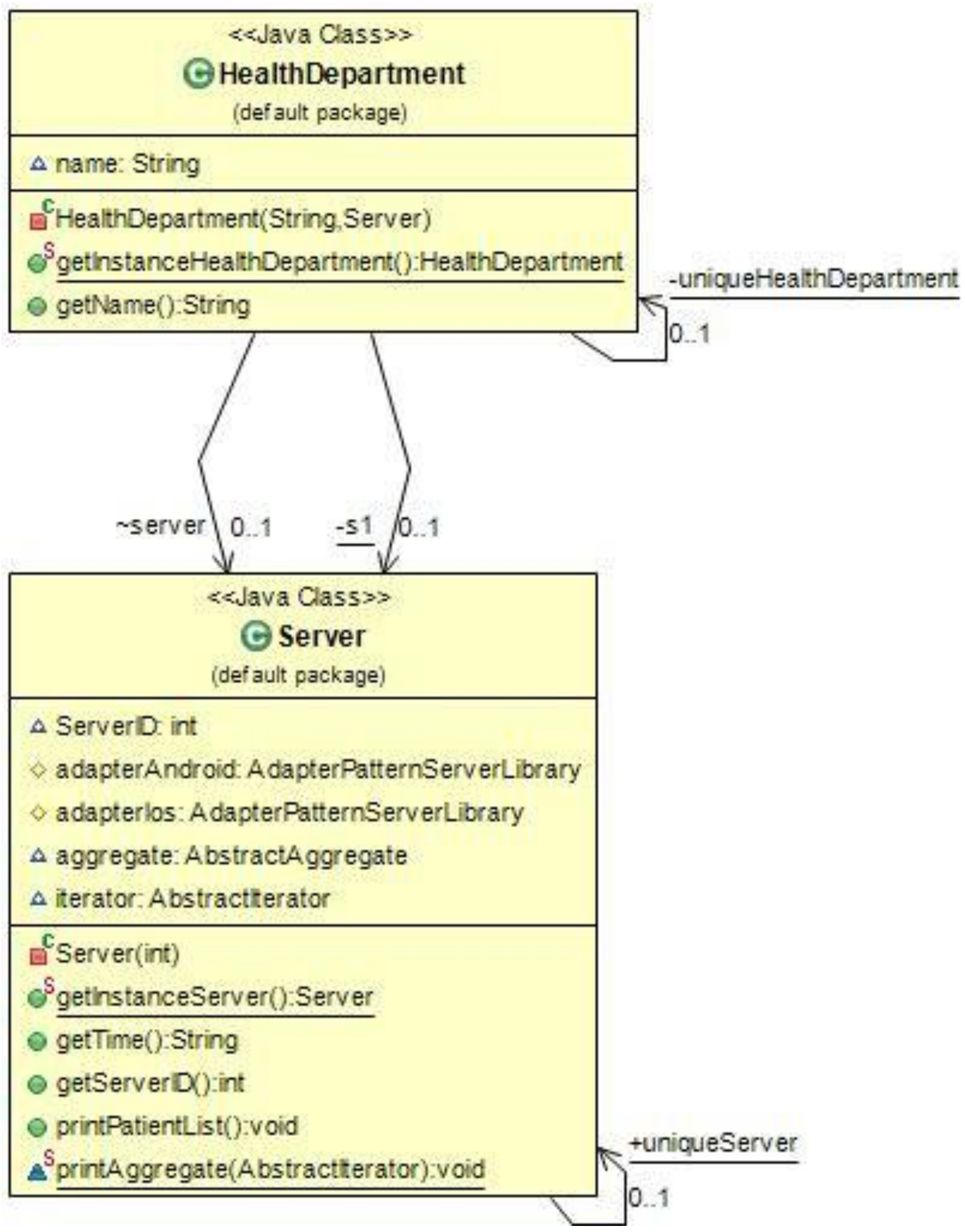
For beginning, we decided to use singleton pattern definitely because the health ministry needs to be one in the project and test use this health ministry. Also, health ministry needs one server to receive data, send data, apply queries. We also provided singleton pattern for server. The lazy instantiation was used in both classes.

Then we noticed that the adapter pattern is useful to handle communication between methods of Android devices, iOS devices and our server. We created two adapters which are one side connected to the server library and other side connected the Android or iOS. We took device type with integer and according to input which comes from user in add part of main loop, directed to its adaptor iOS or Android.

In query part, we assumed that there are two types of query. One of them is server query. We created server query with template design pattern. The server query has three methods in template method. Firstly, age query works and print names and ages of patient between 25 – 60. Secondly, address query works and print addresses of patients with their name. Thirdly, count query works and print count of patients. These methods work sequentially.

Other type of query is patient query. The patients query class has three queries which are location, condition info and last time of update their condition info. Patient query needs ability of multiple or single query in the three queries. We decided that the user can decide these queries with the switch block, but we cannot all implementing in test class again and again. Façade design pattern helps us to making the queries easier to use. We crated firstly façade and interface which has only one method apply query. Queries are implementing the interface according to their queries such as location, condition info or last update time. In test class, the user can access to these queries with one line.

Lastly, we had patients and we can use iterator design pattern to traverse the patients for each query. Before the iterator pattern, we used array list from patients class and we accessed directly. After iterator design pattern, we used patients class as concrete collection and add other iterator design pattern classes such as aggregate, iterator and their concretes.



<<Java Class>>
HealthDepartment
(default package)
name: String
HealthDepartment(String,Server)
getInstanceHealthDepartment():HealthDepartment
getName():String
-uniqueHealthDepartment
0..1
~server
0..1
-s1
0..1
<<Java Class>>
Server
(default package)
ServerID: int
adapterAndroid: AdapterPatternServerLibrary
adapterIos: AdapterPatternServerLibrary
aggregate: AbstractAggregate
iterator: AbstractIterator
Server(int)
getInstanceServer():Server
getTime():String
getServerID():int
printPatientList():void
printAggregate(AbstractIterator):void
+uniqueServer
0..1

Figure 1 - Health Department and Server Singleton Design UML Class Diagram

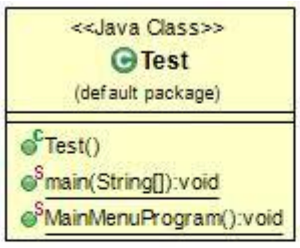


Figure 2 - Test Class UML Class Diagram

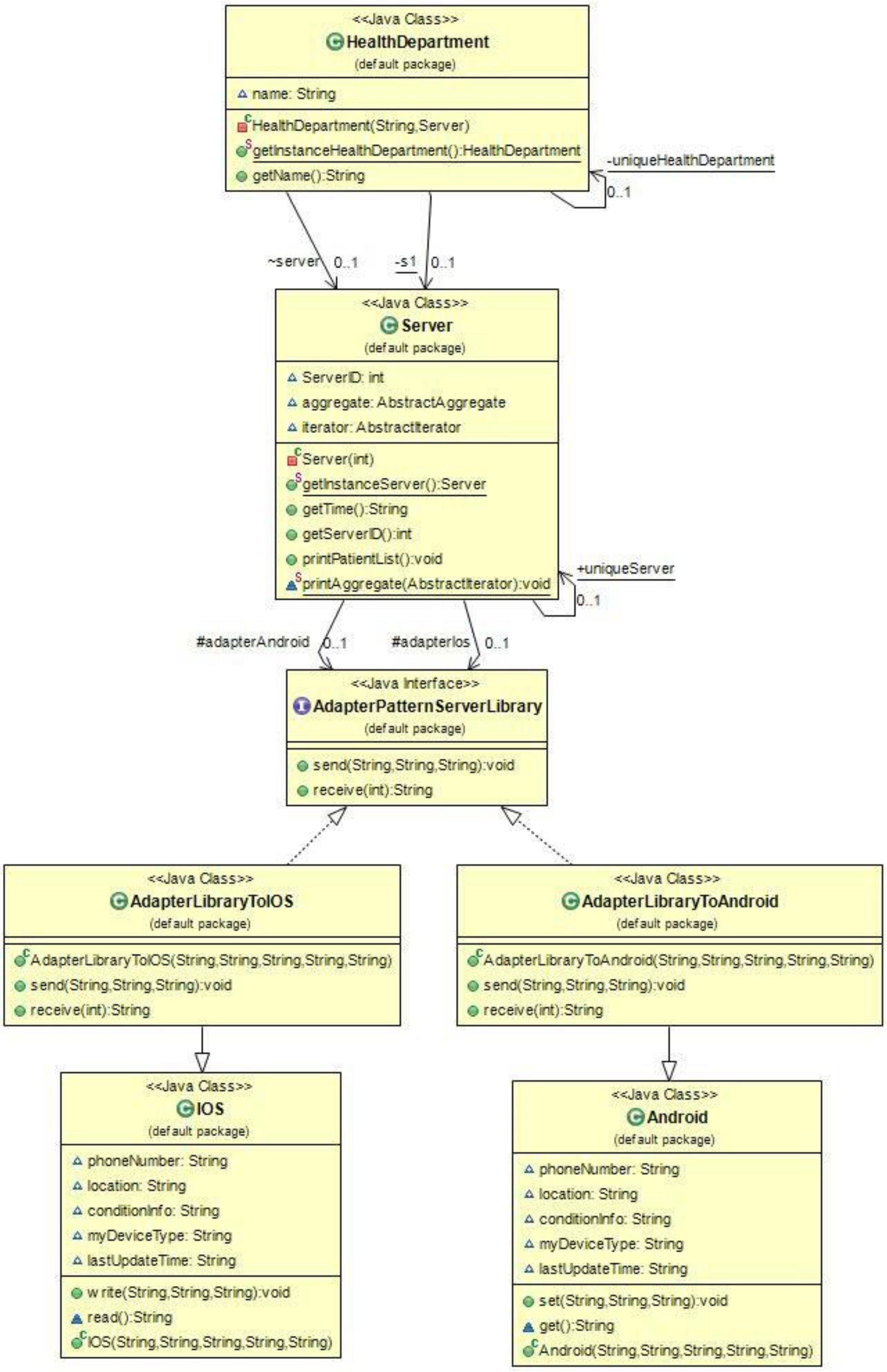


Figure 3 - Adapter Design UML Class Diagram

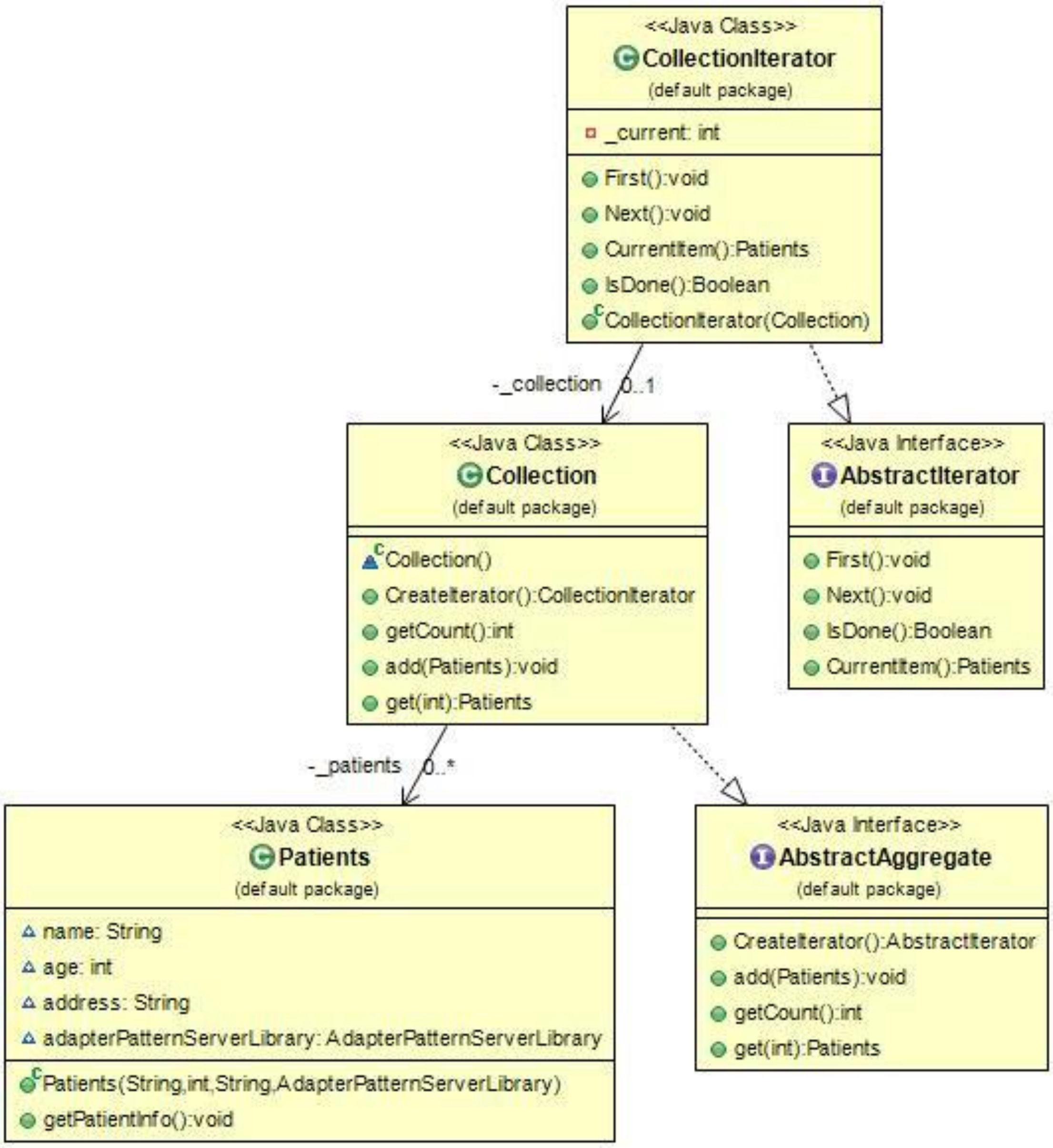


Figure 4 - Iterator Design UML Class Diagram

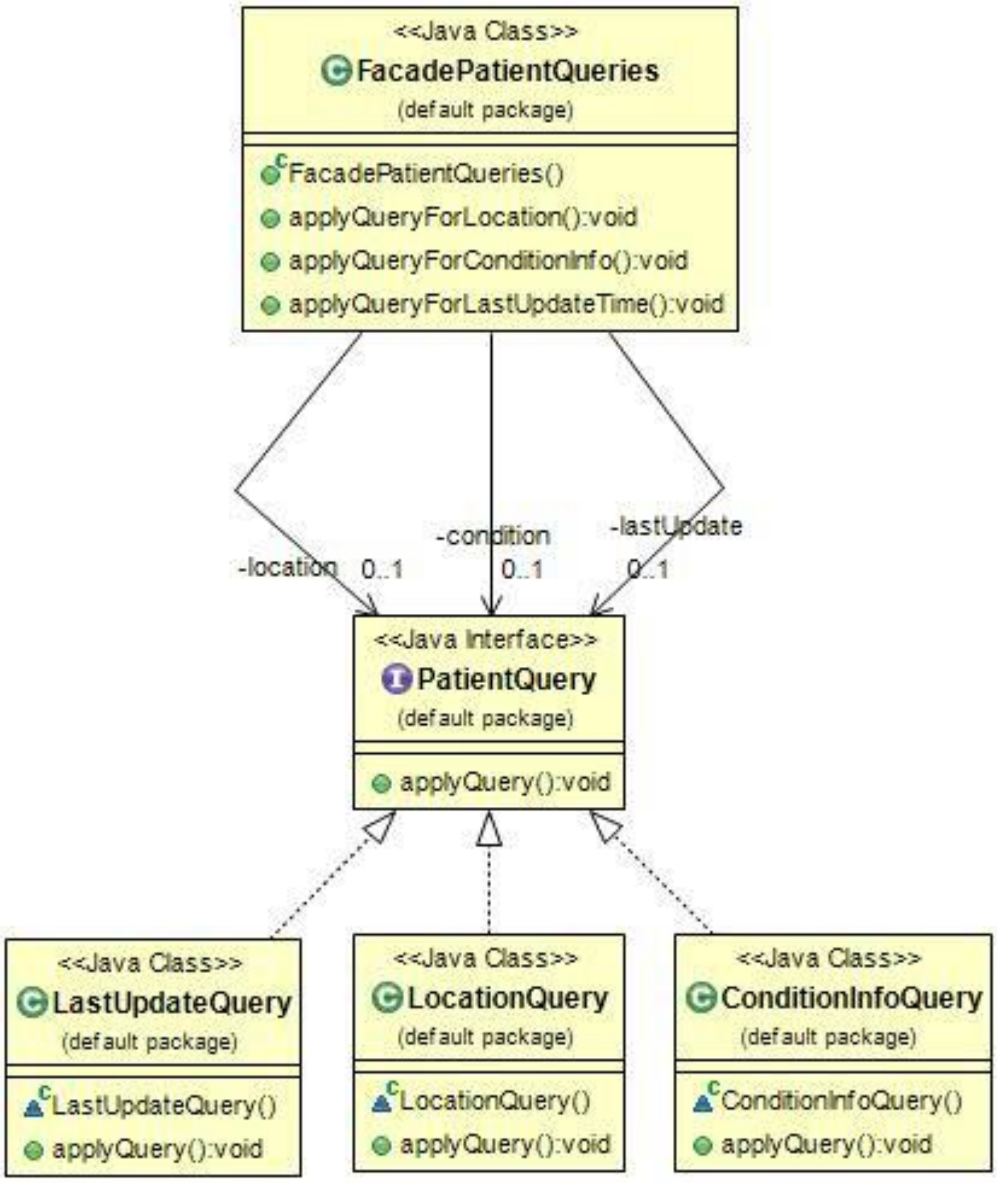


Figure 5 Façade Design UML Diagram

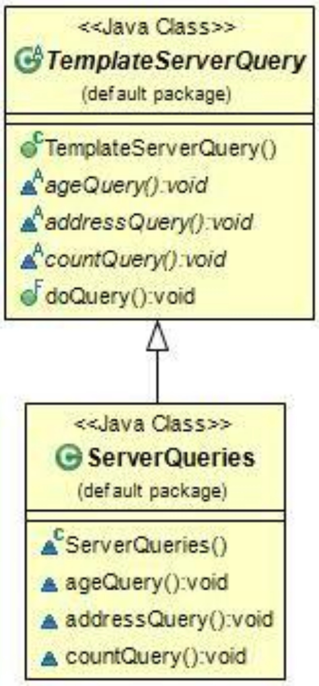


Figure 6 Template Design UML Diagram

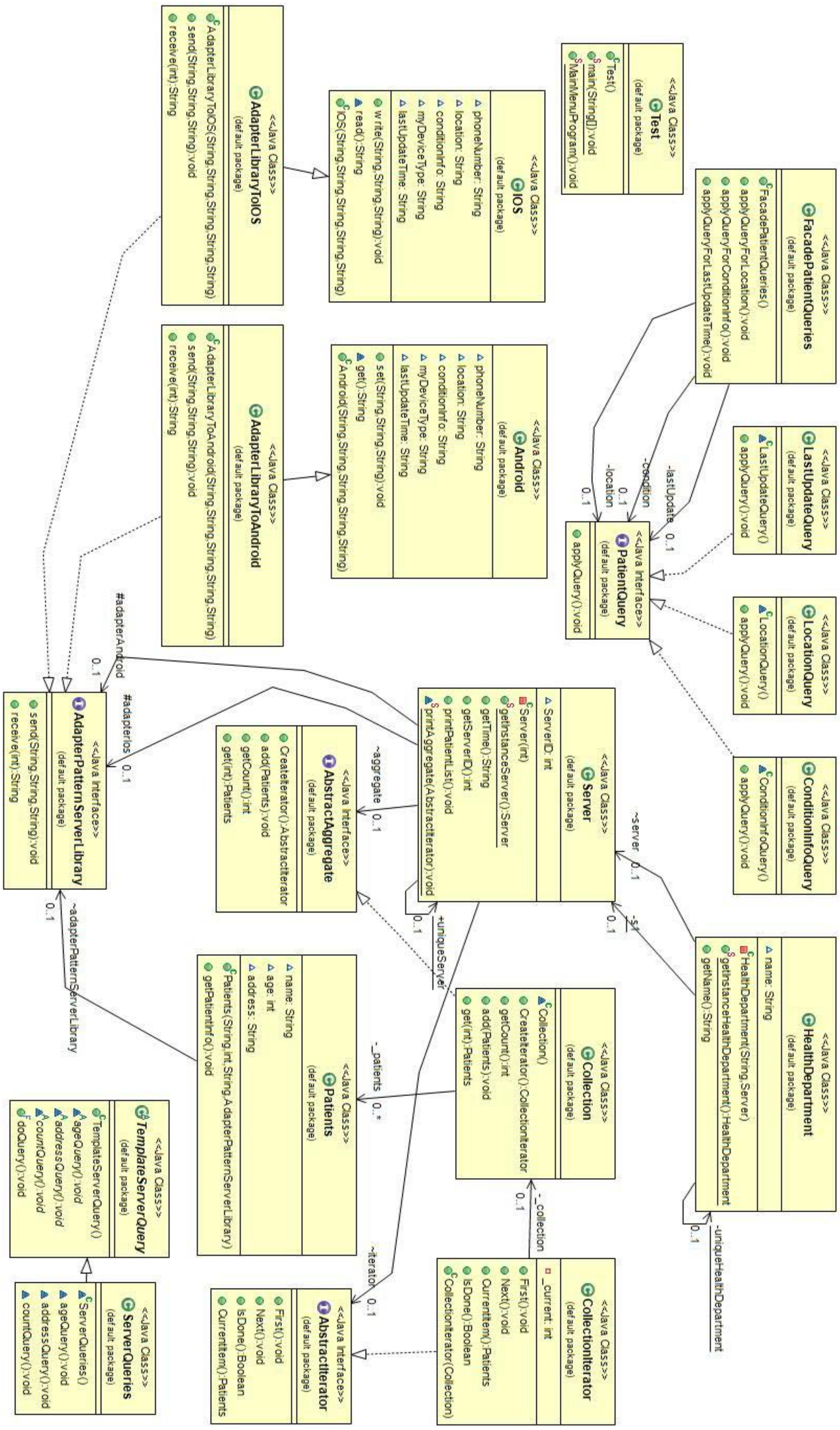


Figure 7 - All lasses UML Class Diagram

In AbstractAggregate java file, AbstractAggregate class is an interface that includes four methods. The CreateIterator and getCount without taking parameter. The get method taking integer and the add method taking patient in Patient class. AbstractIterator class is an interface and has four methods to control collection elements. First, Next, IsDone and CurrentItem without taking parameter. Collection class implements the AbstractAggregate interface. Collection class has private array list. CreateIterator returns new CollectionIterator in here from CollectionIterator class. Implemented method of getCount returns size of patient list. Implemented method of add, adding the patient list taking patient. Implemented method of get returns patients index number in list. CollectionIterator class implements AbstractIterator interface. First method does 0 value of current. Next method increases current value by one. CurrentItem method checks IsDone and if it is false calls the get method inside the collection with current value. IsDone methods checks the current value is bigger or equal than patient list size with getCount method. Patient class has 4 attribute name, age, address and adapterPatternServerLibrary object from its class. Also, the class includes constructor and printPatientInformation method. In printPatientInformation, adapterPatternServerLibrary calls receive with choice parameter. That means print all information belongs to this person.

In AdapterPatternServerLibrary java file, there is one interface in AdapterPatternServerLibrary class. There are two methods which are send and receive to communicate devices. Method of send takes three parameters for send data to devices. Parameters are location, condition and last update time (currently time). In receive method, method takes one parameter to organized coming data. Data comes from adapter according to this parameter value. AdapterLibraryToAndroid class is adapter for Android class and implements adapterPatternServerLibrary (target). Also, the class extends Android class. In adapter class, there is constructor with super function that belongs to Android attributes. The class overrides send method takes three parameters and calls set method from Android class. The receive methods overriding as well as send method, but choice parameter provides to organize the data. If choice parameter is 0, receive method will returns get methods that belongs to Android class to take all device data such as number, location, condition info, last update time and device type. If the parameter is 1 or 2 or 3, receive method returns only one information. We used location, condition info and last update time. In AdapterLibraryToIOS class, the class has same mentality with AdapterLibraryToAndroid, but according to IOS class there are little differences. One of difference is the class extends IOS class. In class, send method calls write method from IOS class and the receive method calls read method from IOS class if choice parameter was 0 as well as in Android adapter class. In IOS class, there are 5 attributes that keeps data for devices. Write method takes 3 parameters from these attributes and changes with these values. Read method prints these all attributes values. Also, the class has own constructor. In Android class, the class is same with IOS class out of set method that takes parameter and change attributes values and get method that prints all attributes values of class. Also, has own constructor.

In FaçadePatientQueries java file, there is an interface that called PatientQuery. The interface has one method applyQuery to override for concrete of subsystem. There are three concrete classes implements PatientQuery in subsystem classes. One of them is LocationQuery. The class has only one method that is overriding applyQuery method by using iterator methods such as First, IsDone, Next and CurrentItem. The method calls receive method with adapter object and choice parameter of receive. The parameter is 1 because of needs to only location information of each patient. In ConditionQuery and LastUpdateQuery works like LocationQuery class with one difference. The difference is choice parameter in receive method inside applyQuery. In ConditionQuery class, calls receive method with 3 for choice parameter as well as LocationQuery. In LastUpdateTime, calls receive method with 2 for choice parameter as well as LocationQuery. The last class of the java file is FaçadePatientQueries. In the class, initialized private three references belong to concrete classes of the interface. The class has constructor. In constructor these references equalized with new objects related with own concrete classes. After that, there are three methods that are applyQueryForLocation, applyQueryForConditionInfo, applyQueryForLastUpdateTime. Each method has one line that is calls applyQuery by using own three attributes.

In TemplateServerQuery java file, there is abstract class that called TemplateServerQuery. In the class, there are three abstract methods that are ageQuery, addressQuery and countQuery. Also, there is template method that called doQuery. In doQuery method, calls ageQuery, addressQuery and countQuery sequentially. ServerQueries extends TemplateServerQuery so overrides these methods inside TemplateServerQuery. Overridden ageQuery methods prints the patients name and their ages between 25 and 60 by using CollectionIterator methods. In overridden addressQuery methods, the method prints the patients name and their addresses. Also, overridden countQuery methods prints patient list length by using aggregate object and getCount method that belongs to Collection class.

In HealthDepartment java file, there are two classes that are Server and HealthDepartment. These classes are cornerstones of project because we passed to classes especially in Test class. In server class, there is one integer attribute for server id. Also initialized adapters, aggregate and iterator in the class. There is private constructor because of singleton design pattern. Static variable uniqueServer initialized with new server object that has 2020 server id in the class. The lazy instantiation used for singleton pattern. Static method getInstanceServer returns uniqueServer if the program needs to server object. Also, there is getTime method that returns current time as string and getServerID method that returns ServerID attribute. Other static printAggregate methods takes AbstractIterator object and calls getPatientInfo methods for each patient by using CollectionIterator methods such that First, Next, IsDone and CurrentItem. Last method is printPatientList in Server class. The method calls printAggregate method by giving iterator object that already created in the class. In HealtDepartment class, the class has name attribute and referenced server in Server class. The class has private constructor because of singleton pattern. Private and static Server object that called s1 referenced to comes from getInstanceServer method in Server class to provide singleton pattern for Server class. Then created new static and private HealthDepartment object that called uniqueHealthDepartment with s1 object (uniqueServer) and Health Department name. Public and static getInstanceHealtDepartment method returns reference of uniqueHealthDepartment. Also, getName method returns name of health department.

In Test class, there is one main menu that is static MainMenuProgram method. The user can display list all patients, add new patient, learn count of patients, update patient's location and condition info with current time, apply server query that age (25-60), addresses and count of patients together and sequentially, apply query location, condition info and last update time one by one or multiple with user wants and quit the program. The method has try-catch-finally blocks to avoid wrong type input while selecting main menu selection. Lastly, the main of the Test class calls only MainMenuProgram method to avoid data losing if the user wrong type input in main menu selection part.