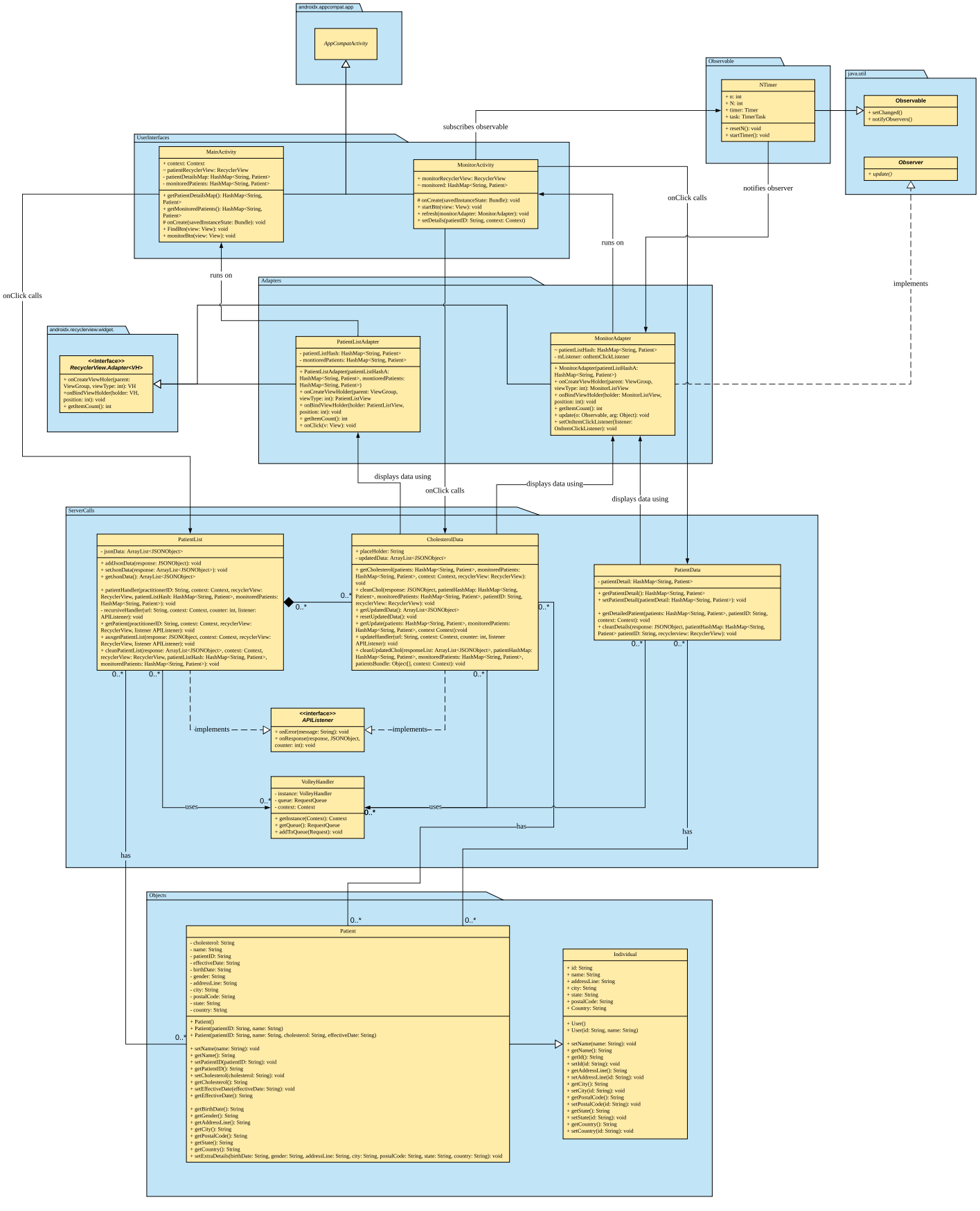
androidx.appcompat.app
AppCompatActivity
Observable
NTimer
+ n: int
+ N: int
+ timer: Timer
+ task: TimerTask
+ resetN(): void
+ startTimer(): void
java.util
Observable
+ setChanged()
+ notifyObservers()
Observer
+ update()
subscribes observable
notifies observer
UserInterfaces
MainActivity
+ context: Context
~ patientRecyclerView: RecyclerView
- patientDetailsMap: HashMap<String, Patient>
- monitoredPatients: HashMap<String, Patient>
+ getPatientDetailsMap(): HashMap<String, Patient>
+ getMonitoredPatients(): HashMap<String, Patient>
# onCreate(savedInstanceState: Bundle): void
+ FindBtn(view: View): void
+ monitorBtn(view: View): void
MonitorActivity
+ monitorRecyclerView: RecyclerView
~ monitored: HashMap<String, Patient>
# onCreate(savedInstanceState: Bundle): void
+ startBtn(view: View): void
+ refresh(monitorAdapter: MonitorAdapter): void
+ setDetails(patientID: String, context: Context)
onClick calls
runs on
runs on
onClick calls
implements
Adapters
androidx.recyclerview.widget
<<interface>>
RecyclerView.Adapter<VH>
+ onCreateViewHolder(parent: ViewGroup, viewType: int): VH
+onBindViewHolder(holder: VH, position: int): void
+ getItemCount(): int
PatientListAdapter
- patientListHash: HashMap<String, Patient>
- monitoredPatients: HashMap<String, Patient>
+ PatientListAdapter(patientListHashA: HashMap<String, Patient>, monticoredPatients: HashMap<String, Patient>)
+ onCreateViewHolder(parent: ViewGroup, viewType: int): PatientListView
+ onBindViewHolder(holder: PatientListView, position: int): void
+ getItemCount(): int
+ onClick(v: View): void
MonitorAdapter
~ patientListHash: HashMap<String, Patient>
- mListener: onItemClickListener
+ MonitorAdapter(patientListHashA: HashMap<String, Patient>)
+ onCreateViewHolder(parent: ViewGroup, viewType: int): MonitorListView
+ onBindViewHolder(holder: MonitorListView, position: int): void
+ getItemCount(): int
+ update(o: Observable, arg: Object): void
+ setOnItemClickListener(listener: OnItemClickListener): void
displays data using
onClick calls
displays data using
displays data using
ServerCalls
PatientList
- jsonData: ArrayList<JSONObject>
+ addJsonData(response: JSONObject): void
+ setJsonData(response: ArrayList<JSONObject>): void
+ getJsonData(): ArrayList<JSONObject>
+ patientHandler(practitionerID: String, context: Context, recyclerView: RecyclerView, patientListHash: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>): void
- recursiveHandler(url: String, context: Context, counter: int, listener: APIListener): void
+ getPatient(practitionerID: String, context: Context, recyclerView: RecyclerView, listener APIListener): void
+ auxgetPatientList(response: JSONObject, context: Context, recyclerView: RecyclerView, listener APIListener): void
+ cleanPatientList(response: ArrayList<JSONObject>, context: Context, recyclerView: RecyclerView, patientListHash: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>): void
CholesterolData
+ placeHolder: String
- updatedData: ArrayList<JSONObject>
+ getCholesterol(patients: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>, context: Context, recyclerView: RecyclerView): void
+ cleanChol(response: JSONObject, patientHashMap: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView): void
+ getUpdatedData(): ArrayList<JSONObject>
+ resetUpdatedData(): void
+ getUpdate(patients: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>, context Context):void
+ updateHandler(url: String, context: Context, counter: int, listener APIListener): void
+ cleanUpdateChol(responseList: ArrayList<JSONObject>, patientHashMap: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>, patientsBundle: Object[], context: Context): void
PatientData
- patientDetail: HashMap<String, Patient>
+ getPatientDetail(): HashMap<String, Patient>
+ setPatientDetail(patientDetail: HashMap<String, Patient>): void
+ getDetailedPatient(patients: HashMap<String, Patient>, patientID: String, context: Context): void
+ cleanDetails(response: JSONObject, patientHashMap: HashMap<String, Patient> patientID: String, recyclerview: RecyclerView): void
0..*
0..*
0..*
0..*
0..*
0..*
0..*
0..*
0..*
0..*
<<interface>>
APIListener
+ onError(message: String): void
+ onResponse(response, JSONObject, counter: int): void
implements
implements
VolleyHandler
- instance: VolleyHandler
- queue: RequestQueue
- context: Context
+ getInstance(Context): Context
+ getQueue(): RequestQueue
+ addToQueue(Request): void
0..*
0..*
uses
uses
has
has
has
Objects
Patient
- cholesterol: String
- name: String
- patientID: String
- effectiveDate: String
- birthDate: String
- gender: String
- addressLine: String
- city: String
- postalCode: String
- state: String
- country: String
+ Patient()
+ Patient(patientID: String, name: String)
+ Patient(patientID: String, name: String, cholesterol: String, effectiveDate: String)
+ setName(name: String): void
+ getName(): String
+ setPatientID(patientID: String): void
+ getPatientID(): String
+ setCholesterol(cholesterol: String): void
+ getCholesterol(): String
+ setEffectiveDate(effectiveDate: String): void
+ getEffectiveDate(): String
+ getBirthDate(): String
+ getGender(): String
+ getAddressLine(): String
+ getCity(): String
+ getPostalCode(): String
+ getState(): String
+ getCountry(): String
+ setExtraDetails(birthDate: String, gender: String, addressLine: String, city: String, postalCode: String, state: String, country: String): void
Individual
+ id: String
+ name: String
+ addressLine: String
+ city: String
+ state: String
+ postalCode: String
+ Country: String
+ User()
+ User(id: String, name: String)
+ setName(name: String): void
+ getName(): String
+ getId(): String
+ setId(id: String): void
+ getAddressLine(): String
+ setAddressLine(id: String): void
+ getCity(): String
+ setCity(id: String): void
+ getPostalCode(): String
+ setPostalCode(id: String): void
+ getState(): String
+ setState(id: String): void
+ getCountry(): String
+ setCountry(id: String): void
0..*
0..*
0..*

Design Rationale

In our class diagram, you can see that all the classes are packaged according to their functionality and dependencies, which aims to apply two package principles, which are the Common Closure Principle and the Common Reuse Principle.

The Common Closure Principle that states that we should group together classes that are likely to change together for some reason. We can see this here as the classes VolleyHandler, APIListener, CholesterolData, PatientData and PatientList are grouped together under the same package called ServerCalls. These classes mainly facilitate and make the API calls, clean the data and pass the data to the adapters. These classes are dependent on each to a certain extent other and flow in somewhat of a sequence. Hence, if a change is needed, a change in most of the classes is likely. By grouping these classes together in a package, we are able to minimize the number of packages that must be re-released when a change is made.

Next, we can also observe the Common Reuse Principle. An example of this is the Adapters package which contains the MonitorAdapter and PatientListAdapter class. These classes are reused again when new data from the api is needed to be displayed. For example, the app has the N-seconds refresh function where we get the latest cholesterol values of the monitored patients. When the latest data is obtained and cleaned, MonitorAdapter is reused to display the updated results.

Another design principle is the Open-Closed principle. Besides from our classes inheriting from others, some classes also implement interfaces where its methods are overwritten when needed. An example of this is when the PatientList and CholesterolData classes implement the APIListener interface. This is needed as Volley API requests are asynchronous, hence we need a callback so that we can implement certain methods only when we get a response from the API. The Open-closed principle allows us to extend our classes and add functionality when needed.

One of the design patterns used here is an Observable. As shown in the class diagram, the NTimer class inherits the Observable class, while the MonitorAdapter implements the Observer class. This allows us to notify the MonitorAdapter (observer) when N seconds have passed. Since Observer is an abstract class, we can easily extend and create multiple observers, making use of the Open-Close principle and Liskov Substitution principles. The Liskov Substitution principle says that all subclasses should also be subtypes. This is true for NTimer which is a subclass of Observable. Currently there is only a need for one observable, but if we were to add new functionalities, we might need more observables, or in other words, more subtype of an Observable class. Another example for this is how the Patient class extends from the Individual class. In the future, we might also need different types of Individual.