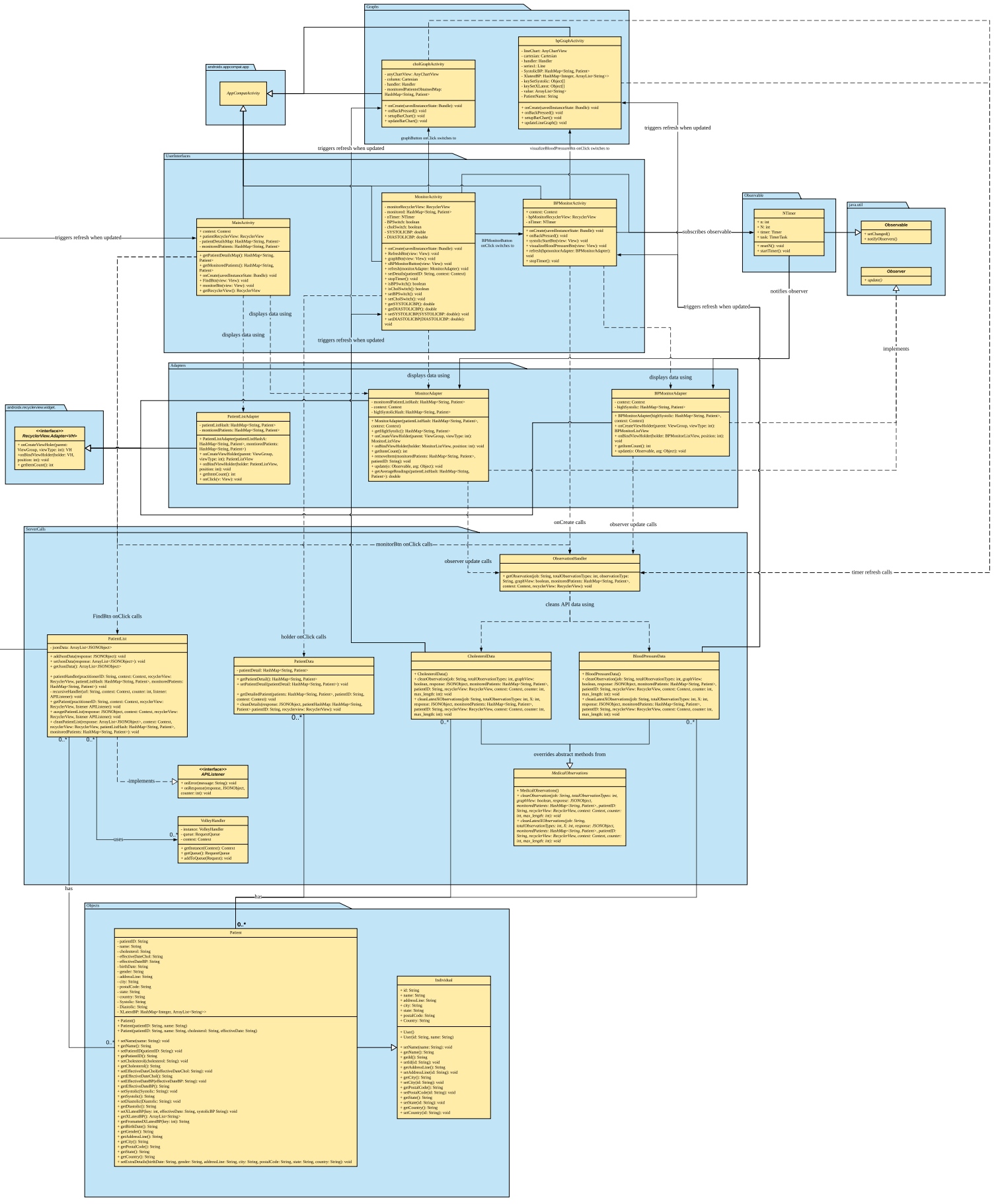
Graphs
androidx.appcompat.app
AppCompatActivity
cholGraphActivity
- anyChartView: AnyChartView
- column: Cartesian
- handler: Handler
- monitoredPatientsObtainedMap: HashMap<String, Patient>
+ onCreate(savedInstanceState: Bundle): void
+ onBackPressed(): void
+ setupBarChart(): void
+ updateBarChart(): void
bpGraphActivity
- lineChart: AnyChartView
- cartesian: Cartesian
- handler: Handler
- series1: Line
- SystolicBP: HashMap<String, Patient>
- XlatestBP: HashMap<Integer, ArrayList<String>>
- keySetSystolic: Object[]
- keySetXLatest: Object[]
- value: ArrayList<String>
- PatientName: String
+ onCreate(savedInstanceState: Bundle): void
+ onBackPressed(): void
+ setupBarChart(): void
+ updateLineGraph(): void
graphButton onClick switches to
visualizeBloodPressureBtn onClick switches to
triggers refresh when updated
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
+ getRecyclerView(): RecyclerView
triggers refresh when updated
displays data using
MonitorActivity
- monitorRecyclerView: RecyclerView
- monitored: HashMap<String, Patient>
- tTimer: NTimer
- BPSwitch: boolean
- cholSwitch: boolean
- SYSTOLICBP: double
- DIASTOLICBP: double
+ onCreate(savedInstanceState: Bundle): void
+ RefreshBtn(view: View): void
+ graphBtn(view: View): void
+ tdPMonitorButton(view: View): void
+ refresh(monitorAdapter: MonitorAdapter): void
+ setDetail(patientID: String, context: Context)
+ stopTimer(): void
+ isBPSwitch(): boolean
+ isCholSwitch(): boolean
+ setBPSwitch(): void
+ setCholSwitch(): void
+ getSYSTOLICBP(): double
+ getDIASTOLICBP(): double
+ setSYSTOLICBP(SYSTOLICBP: double): void
+ setDIASTOLICBP(DIASTOLICBP: double): void
BPMonitorButton onClick switches to
BPMonitorActivity
+ context: Context
- bpMonitorRecyclerView: RecyclerView
- tTimer: NTimer
+ onCreate(savedInstanceState: Bundle): void
+ onBackPressed(): void
+ systolicStartBtn(view: View): void
+ visualizeBloodPressureBtn(view: View): void
+ refresh(bpmonitorAdapter: BPMonitorAdapter): void
+ stopTimer(): void
subscribes observable
Observable
NTimer
+ tc: int
+ N: int
+ timer: Timer
+ task: TimerTask
+ restartN(): void
+ startTimer(): void
java.util
Observable
+ setChanged()
+ notifyObservers()
Observer
+ update()
notifies observer
implements
Adapters
PatientListAdapter
- patientListHash: HashMap<String, Patient>
- monitoredPatients: HashMap<String, Patient>
+ PatientListAdapter(patientListHashA: HashMap<String, Patient>, monitoredPatients: HashMap<String, Patient>)
+ onCreateViewHolder(parent: ViewGroup, viewType: int): PatientListView
+ onBindViewHolder(holder: PatientListView, position: int): void
+ getItemCount(): int
+ onClick(v: View): void
displays data using
MonitorAdapter
- monitoredPatientListHash: HashMap<String, Patient>
- context: Context
- highSystolicHash: HashMap<String, Patient>
+ MonitorAdapter(patientListHash: HashMap<String, Patient>, context: Context)
+ getHighSystolic(): HashMap<String, Patient>
+ onCreateViewHolder(parent: ViewGroup, viewType: int): MonitorListView
+ onBindViewHolder(holder: MonitorListView, position: int): void
+ getItemCount(): int
+ removeItem(monitoredPatients: HashMap<String, Patient>, patientID: String): void
+ update(o: Observable, arg: Object): void
+ getAverageReadings(patientListHash: HashMap<String, Patient>): double
BPMonitorAdapter
- context: Context
- highSystolic: HashMap<String, Patient>
+ BPMonitorAdapter(highSystolic: HashMap<String, Patient>, context: Context)
+ onCreateViewHolder(parent: ViewGroup, viewType: int): BPMonitorListView
+ onBindViewHolder(holder: BPMonitorListView, position: int): void
+ getItemCount(): int
+ update(o: Observable, arg: Object): void
displays data using
androidx.recyclerview.widget
<<interface>>
RecyclerView.Adapter<VH>
+ onCreateViewHolder(parent: ViewGroup, viewType: int): VH
+onBindViewHolder(holder: VH, position: int): void
+ getItemCount(): int
ServerCalls
onCreate calls
observer update calls
monitorBtn onClick calls
observer update calls
ObservationHandler
+ getObservation(job: String, totalObservationTypes: int, observationType: String, graphView: boolean, monitoredPatients: HashMap<String, Patient>, context: Context, recyclerView: RecyclerView): void
timer refresh calls
cleans API data using
FindBtn onClick calls
holder onClick calls
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
+ receiveHandler(url: String, context: Context, counter: int, listener: APIListener): void
+ getPatient(practitionerID: String, context: Context, recyclerView: RecyclerView, listener: APIListener): void
+ assignPatient.ist(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
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
CholesterolData
+ CholesterolData()
+ cleanObservation(job: String, totalObservationTypes: int, graphView: boolean, response: JSONObject, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView, context: Context, counter: int, max_length: int): void
+ cleanLatestXObservations(job: String, totalObservationTypes: int, X: int, response: JSONObject, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView, context: Context, counter: int, max_length: int): void
BloodPressureData
+ BloodPressureData()
+ cleanObservation(job: String, totalObservationTypes: int, graphView: boolean, response: JSONObject, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView, context: Context, counter: int, max_length: int): void
+ cleanLatestXObservations(job: String, totalObservationTypes: int, X: int, response: JSONObject, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView, context: Context, counter: int, max_length: int): void
overrides abstract methods from
MedicalObservations
+ MedicalObservations()
+ cleanObservation(job: String, totalObservationTypes: int, graphView: boolean, response: JSONObject, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView, context: Context, counter: int, max_length: int): void
+ cleanLatestXObservations(job: String, totalObservationTypes: int, X: int, response: JSONObject, monitoredPatients: HashMap<String, Patient>, patientID: String, recyclerView: RecyclerView, context: Context, counter: int, max_length: int): void
<<interface>>
APIListenor
+ onError(message: String): void
+ onResponse(response, JSONObject, counter: int): void
implements
VolleyHandler
- instance: VolleyHandler
- queue: RequestQueue
- context: Context
+ getInstance(Context): Context
+ getQueue(): RequestQueue
+ addToQueue(Request): void
uses
has
Objects
Patient
- patientID: String
- name: String
- cholesterol: String
- effectiveDateChol: String
- effectiveDateBP: String
- birthDate: String
- gender: String
- addressLine: String
- city: String
- postalCode: String
- state: String
- country: String
- Systolic: String
- Diastolic: String
- XLatestBP: HashMap<Integer, ArrayList<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
+ setEffectiveDateChol(effectiveDateChol: String): void
+ getEffectiveDateChol(): String
+ setEffectiveDateBP(effectiveDateBP: String): void
+ getEffectiveDateBP(): String
+ setSystolic(Systolic: String): void
+ getSystolic(): String
+ setDiastolic(Diastolic: String): void
+ getDiastolic(): String
+ setXLatestBP(key: int, effectiveDate: String, systolicBP String): void
+ getXLatestBP(): ArrayList<String>
+ getFormattedXLatestBP(key: int): 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..*
1

Design Rationale

Similar to our previous design, all of our classes are packaged according to their functionality and dependencies. This allows us to apply the Common Closure Principle and Common Reuse Principle. Other design principles or patterns include Open-Closed Principle, Liskov Substitution Principle, Observable Pattern, etc. These principles and patterns can be seen in both the classes from Assignment 2 and the new classes added in for Assignment 3. Below, we explain more about the significant changes and differences in the current system.

In our previous design from Assignment 2, we had separate methods for getting the cholesterol values for the first time and when an update is needed. In the current design, we have refactored and successfully reduced a large amount of repeated code. In our refactoring process, we used many different refactoring techniques such as renaming methods and variables so that they are consistent with their functionality. We also implemented a combination of the move method and inlining method. As the CholesterolData class had multiple methods, we moved them out into separate classes and removed the unnecessary ones, allowing us to refactor and reduce the whole process to a minimal number of methods.

The main result of this refactoring is the ObservationHandler class which is made up of one method that calls the API to get observations for both cholesterol and blood pressure. The response from the API will have to be cleaned, and this is different for cholesterol and blood pressure. In order to do this, we used the Open-Closed Principle and Liskov Substitution Principle. We created an abstract class called MedicalObservations with methods to be overridden. The CholesterolData and BloodPressureData inherits from this class and overrides its methods with its own implementations. These two classes are now subtypes of MedicalObservations, and the MedicalObservations class is available to add additional abstract methods as well as have more subclasses.

Besides that, we have also implemented a similar variation of the Model-View-Controller (MVC) pattern. This design pattern can be seen throughout our system whenever an activity class needs to display cholesterol or blood pressure values. In our design, the activity classes will act as the controller. For example, when the monitor button is clicked in the MainActivity class, methods in the ObservationHandler and CholesterolData class will be called to get the cholesterol values. This is the Controller notifying the Model. Once the latest cholesterol values have been obtained, the CholesterolData class will call a method in the MainActivity class to display these values using the MonitorAdapter class. The difference here is that the View gets the latest values from the Controller instead of the Model, which means the View will only be updated when the Model is updated, and not before.