GC02 Team 7 Project Report
MINAP Mobile Application

Amer Kamil Zainal Abidin, Ke Wei, Marco Corrales Estrada

January 21, 2014

Contents

1 Background

1.1 MINAP/NICOR

The Myocardial Ischaemia National Audit Project (MINAP) investigates the management of heart attacks. Participating hospitals and ambulance services are provided with a record of their quality of management. These records are then compared to national and international clinical standards. The National Institute of Cardiovascular Outcomes Research (NICOR) is a partnership of clinicians, IT experts, analysts, academics and managers. Its mission is to "provide information to improve heart disease patients' quality of care and outcomes". Using different audits, MINAP being one of them, NICOR aims to provide the NHS, government and regulatory bodies real world data to be used in patient care [1][2].

1.2 Web Application

Participating hospitals are requested to enter all patients with suspected heart attack into MINAP's central database system. Data are collected by nurses and other clinical audit staff and entered in a dedicated web application. This web application collects all patient data, from demographics to diagnosis and release information. As it stands, the web application contains a 130 field dataset that is updated every two years (latest revision, June 2013); it is currently only available to the web browser Internet Explorer versions 9 or below. The data is then validated against a set of rules which include, but are not limited to, value range checks, date range checks, data type checks and encryption rules. Once completed, records are stored on MINAP's Domino server[3].

1.3 Client

The project's primary point of contact is Lucia Gavalova, Project Manager for the MINAP project, who we will refer to as the 'client' throughout this document (unless stated otherwise). Sue Manuel, MINAP's web app developer has also been a key stakeholder throughout the course of this project.

---

[1]MINAP Reports

[2]NICOR Website

[3]IBM Domino is used for enterprise collaboration, and is used primarily for its Lotus Notes capabilities in this instance

2 Requirements

The following requirements have been formulated from a series of client meetings and e-mail communication, available in brief in appendix B.

2.1 Overview

Patient data record creation is currently restricted to when the nurses and other clinical audit staff are sitting in front of a computer. This would often be quite some time after a patient has been identified as being eligible for being entered into the MINAP dataset. A reduced-function mobile solution of the current web app was proposed by our client as a means of starting a patient record in MINAP's data set, allowing for prospective data collection. Completion of the record is expected to be done later on using the web app (using a full sized keyboard and screen).

2.2 Use Cases

The system's use case we are concerned with for our app can be modelled by the following table. Although the user-facing view of the system is fairly simple, most of the application logic lies in the interdependency and validation checks of the values entered by the user.

A nurse or clinical audit staff wants to create a record
She first needs to login to the server with her credentials
She then selects an option to create a new record
She then is brought to a page to enter in her initial diagnosis
If she doesn't understand a field, she can click on a button for help
She then enters the patient details into a data input field
If the data entered is invalid, a pop-up will appear, prompting her for correction
She can then navigate through each page using 'next' buttons, or through a navigation map
Throughout this process, she will save the data to the server by clicking a save button
Once complete, she logs out of the system

2.3 Functional Requirements

2.3.1 User Authentication and Security

As with the web application, the user must login to the central server prior to using the mobile app for data input. For security purposes, the user authentication details should not be held on the device beyond the initial authentication phase (or instead, use tokens), and all sensitive data retrieved from the server and created on the device should be destroyed at the end of a user session.

2.3.2 Data Collection

In order to create a patient record, the app needs to provide the user with forms for data entry. Depending on the data fields, certain format restrictions are imposed, for example: date and time related fields are in the dd/mm/yy hh:mm format; combo boxes; and numeric only data fields. Data fields should be divided into sections (pages) based on the current web app structure for familiarity. Data entered in by the user should be stored locally for the duration of the session, and ideally sent to a server once validated[4].

2.3.3 Data Validation

Validation currently occurs on the web app automatically at the point of exiting a data input object, page, or on saving. All 130 fields within the dataset must pass at least one validation rule. Certain fields will then either trigger additional validation rules involving other fields, default values to other fields, and/or open new sections of data collection to be filled out. Our app should follow this behaviour (with a subset of data) to ensure data integrity.

2.3.4 Data Submission

Once all mandatory data is entered and all validations rules have been passed, the user will be able to store the local record to a SQLite database. Due to time constraints, the team was unable to test calls to a web service of our own design as agreed with the client (20/11/13 Meeting). However, the SQLite database should prove to be flexible if used in future implementations as it is self-contained and follows the dataset's specifications.

---

[4]See Data Submission

2.3.5 Mobile Usability

Since the app will be running on smart phones (our target platform), factors such as the screen size, touch navigation, and use of an on-screen keyboard have to be taken into account during the user interface design phase. Certain features in the current web app (e.g. navigation and data submission) are not intuitive and need to be rethought for mobile use, but should still retain the familiarity of the existing interface.

2.4 Architectural Requirements

The core purpose of our mobile application is to provide proof of concept for a solution what would offer a starting point for record creation. To that end, we came to a mutual agreement with the client that the original dataset be reduced from 130 fields. In exchange for reducing the volume of data, certain architectural requirements were requested so as to create a framework for future work to build upon (e.g. porting the application to other platforms).

2.4.1 Web Services

Although we have reached an agreement that we will not actually need to touch the live Domino Server, there are certain functions that our application will have to implement to ensure future compatibility with MINAP's technology stack (IBM Lotus Notes on Domino Server 8.5.3). Processes such as how data will be retrieved from the server and authentication processes should be modelled into the system as stubs that can be easily implemented at a later stage in the development of the app. Further research into the topic of communication with the server from other services was also deemed useful, since the client was unable to give a clear answer as to how this could be done.

3 Target Mobile Platform

3.1 Client Preference

3.2 User Survey

THIS SECTION IS TO BE REWRITTEN

Our client has been unsure about the preferred target platform of the audience of our mobile app. To help better inform this decision, we have designed a user device survey which the client has agreed to running over the period of a few months, with a suggested completion date within 3 weeks of rolling out. The primary purpose of pushing out this survey is to better inform decisions of what secondary platforms should be supported to increase mobile app uptake by hospital staff. Despite this, due to the short time-frame imposed on us by GC01, we have mutually agreed to target Android mobile phones as the primary platform. More details regarding the device survey can be found in the Appendix.

THIS SECTION IS TO BE REWRITTEN

4 User Interface

4.1 Preliminary UI Design

As part of ensuring our understanding of the requirements were in line with the client's expectations, we sketched out a preliminary user interface (UI) and presented our ideas to our client. The order in which each page was presented is indicated by red arrows. A summary of the key points relating to UI during our conversation are included below.

4.1.1 Initial User Interface Feedback

Client response:

- happy with proposal of similar design to web app

- current (web app) record search criteria is non-specifiable (simple text match); this behaviour is sufficient for the mobile app

- request for an application tutorial for "non-techie" users

- navigation menu: a simplified version of the map could be used for mobile

- an 'auto-save' feature would be useful ("like Microsoft Word") in case of battery loss, device breakage, or more pressing matters to attend to (i.e. emergency)

4.1.2 Commentary and Response

We have accepted the feedback from the client as being in line with our initial requirements. However, the request for an auto-save feature in the app would violate the privacy requirements of having volatile local data. A way to mitigate this would be to save the record to a 'draft' database when auto-saving (i.e. "submitting" a record without performing validation-on-send), but is a low-priority feature at this point.

4.1.3 UI Tweaking

After taking these comments on board, we have more formally modelled the user's flow of use through the app, mapping each UI element to Java's graphical implementation classes. It is worth noting that the UI is still in its initial stages, pending further adaptation for mobile use.

4.2 Android UI Design

5 Software Design

5.1 Systems Design

5.2 Designing for Android

5.3 Data Model & Validation

In order to capture the functional requirements previously explained, three major steps were to be completed for the inner workings of the mobile application.

5.3.1 Designing the patient singleton

The Patient class would, in essence, capture the entirety of the dataset for this project. For security reasons, the team agreed with the client (5/12/13 Meeting) that the application should not hold more than one record per session. As such, the patient class was deemed fit to be designed as a Singleton. A call to a Patient object will return either an existent instance of the class or create one if such instance does not exist. Furthermore, the clone method of the Patient class has been overridden to throw a CloneNotSupported exception in order prevent stray clones from being created. All fields from the dataset are then declared as part of the patient class.

5.3.2 Designing and implementing field classes and the Value superclass

As previously mentioned, all fields must pass at least one validation rule before entering long term storage, either to the local database or MINAP's server. Although some fields shared similar validation rules, such as date fields needing a certain display format, the usage of a super class was minimized due to the volatility of the dataset itself. Due to this, our superclass, Value, only encapsulates variables that the team decided may not change as frequently as others; namely the field number within the dataset and the description of said field. These two values are used as tooltips on both the existing web application as well as our GUI. All field classes have been designed to be independent of each other.

Each class contains its own data validation rules. The MINAP dataset and sanity checks files provided by the client proved essential in deciding what data types to use for each field. Length check validations were handled by declaring a private short variable that would hold said length value. Date and time check validations were slightly trickier as they required the use of Date methods as well as importing the SimpleDateFormat library to better implement comparisons. Alphanumeric checks would need to be compared against regular expressions.

The getter methods would take an appropriate typed value as a parameter and return boolean true if said parameter passed the field's validation rules. An if statement was the most appropriate structure to check a parameter's value. Strings would need to clear regular expressions; Dates, Integers and Doubles have ranges they must fall within; to capture the drop down list or radio button behavior seen in the web application, classes with switch statements were implemented that both checked the selection for validity and set the fields "Long Code" variable to the appropriate value.

Setter methods, for reasons that will become clearer in the next session, would return the data necessary as a String in the needed display format.

5.3.3   Designing and implementing the DB

The SQLite database used in this project uses no third party libraries. It was clear that the less dependencies present in our code, the easier it may be to fully implement the dataset and network capabilities further down the road. As this approach was chosen, the implementation of the database was more time consuming than expected. The final result, however, was well worth it.

The database itself consists of only one table, one record long. The DBAdapter class starts by declaring the field names to be used, each logical "page" declaration finished by declaring an array of strings that can hold all column names to a particular page. The concatenateArrays method can then declare a larger array that will hold all columns that were previously declared, this method can and indeed is, then be used when a SELECT * statement needs to be used on the database. The database is then created by assigning it a name and a tag that will be used when updating the schema. After assigning the table a name, a String is declared that holds the SQL statement to create the database.

All of the database logic is held from the declaration of the DatabaseHelper class down. Methods to open and close the database are followed by methods that insert information to a set page, updating information in said pages. Special attention must be given to the insertPatient method as it represents the Patient Info page on the UI and is the only way to actually insert one more record onto the database. The DBAdapter class ends with the implementation of a method that will, due to the restriction of one record per session, destroy all data; and a method that will select all data from the record.

The columns on our patient table have all been set to be Text. This decision facilitates writing from the field classes getter methods without having to worry about server format compatibility. This decision, as well as the decision to only hold one record, can be easily expanded and corrected if necessary.

6 Testing & Analytics

7   Application Limitations

While this application does fulfil our client's requests, it is far from a finished product that could be implemented in the workplace. In order to provide a complete solution, our application would need to implement all missing fields from the dataset. These fields would then need to be reflected on the UI. However, the most obvious limitation of our solution is, without a doubt, the lack of network connectivity.

Our main priority during the second half of this project was to produce something tangible. To have something that our client could hold in her hands and, not to put it lightly, play with. To that end, and with her consent, networking research was put on the back burner in favour of delivering a functioning application.

Still, the need for a further study onto the networking technologies used at MINAP was necessary. If not for our project, for future hands to grapple with.

8 Client Feedback

9 Future Plans

9.1 Use of XML/Portable Data Files

The application rules are currently defined by a series of data fields stored across two excel spreadsheets. From what we understand (20/11/13 Meeting), these rules are implemented in client-side Javascript for the current web app. Ideally, these rules would need to be stored in a standard machine-readable format which can be easily translated to validation code in any programming language.

10 Project Management

10.1 Work Packages

10.2 Gantt Chart

		2013																											
		November																											
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Requirements Gathering

GC01 Milestone 1

InitialDevelopment

2013																														
December																														
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

GC01 Milestone 2 ◆ (20)

2014																											
January																											
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

GC02 Final Milestone ◆

11 Teamwork

11.1 Team Roles

Team roles were initially assigned based on the recommended list provided in the GC01 document requirements. These team roles were useful as a set of guidelines leading up to the first project milestone, but we quickly found that the assignment of multiple formal roles to team members was unrealistic for such a fast-paced project with team members at various skill levels. This meant that roles previously assigned to team members were not always reflective of their current work package, as the rest of the team pitched in to complete critical unfinished packages. As such, only the roles which the team member was most active in are tabulated below:

Name	Active Roles
Marco **David** Corrales (Secondary Team Lead)	Secondary Developer Data/Database Designer
Amer **Kamil** Zainal (Primary Team Lead)	Primary Developer Systems Designer & Analyst Documentation Lead
Ke Wei	Networking Researcher

Table 1: Team Member Active Roles

11.2 Teamwork Commentary

11.2.1 Team Workflow

Trello

Trello[5] has been used extensively by the team for information organisation. Agendas for the next meeting are typically decided at the end of each meeting, with unfinished items of the day deferred; ensuring that each agenda is a summary of what was achieved at each meeting. To supplement this, any details (e.g. comments, photos, resources) related to the meeting are included in a separate section with a corresponding date.
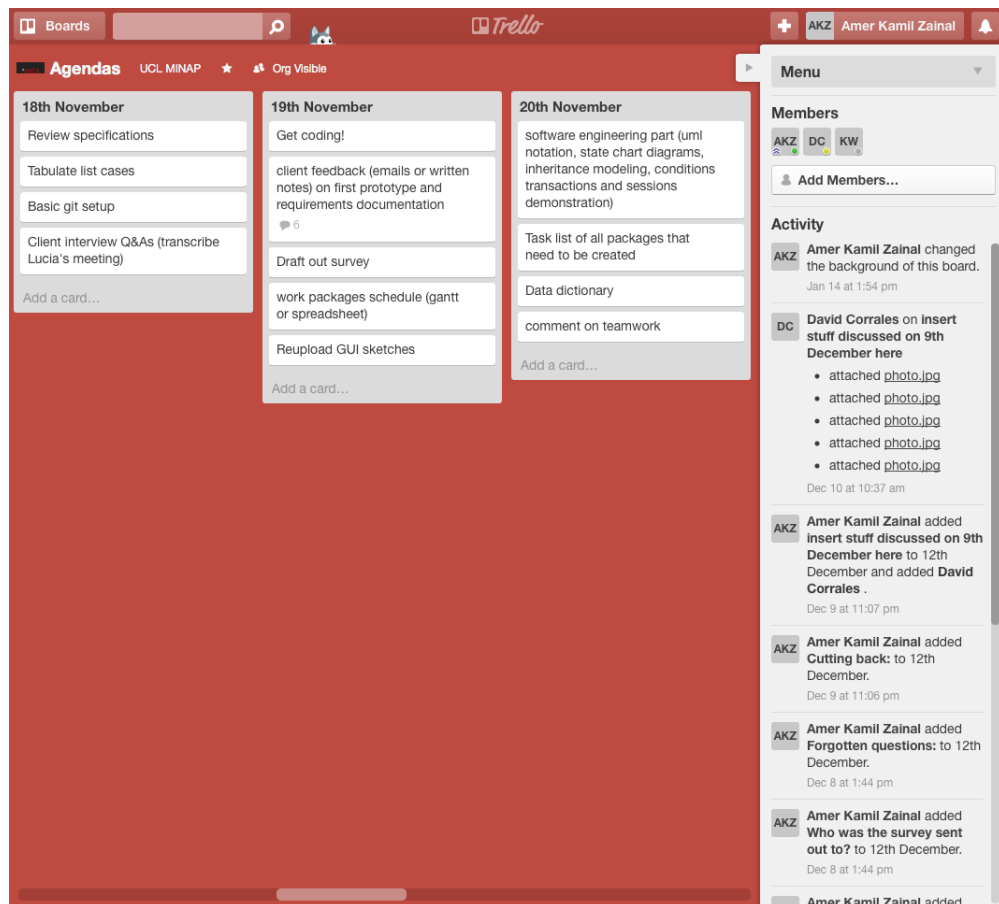

Figure 1: Snapshot of a board in the team's Trello

---

[5]Information disclosed on Trello are private to the team

GitHub

To further consolidate our memory of team progress, David updated a blog (http://akz08.github.io/minap-mobile/) on GitHub Pages using Octopress, summarising our meeting outcomes up to our first academic milestone. Due to the public nature of blogs, we initially ensured to omit any client identifying information to avoid any potential privacy problems, however we later decided that this was unnecessarily restrictive. This openness of communication was also necessary for us to write about our design process for the HCI requirement of our blog. All of the blog posts created are tagged with the author's name for attribution.

For source code collaboration, we used a private GitHub repository which was worked on by all members of the team. GitHub's repository wiki functionality was used for collaboration of more long-form documentation (e.g. writing this report) and structured ordering of resources. Due to the varying experience levels with git, the team mostly made use of simple git branches for each person. A full copy of the git repository is included in the project source code submission for closer inspection, contributing to the large file size.
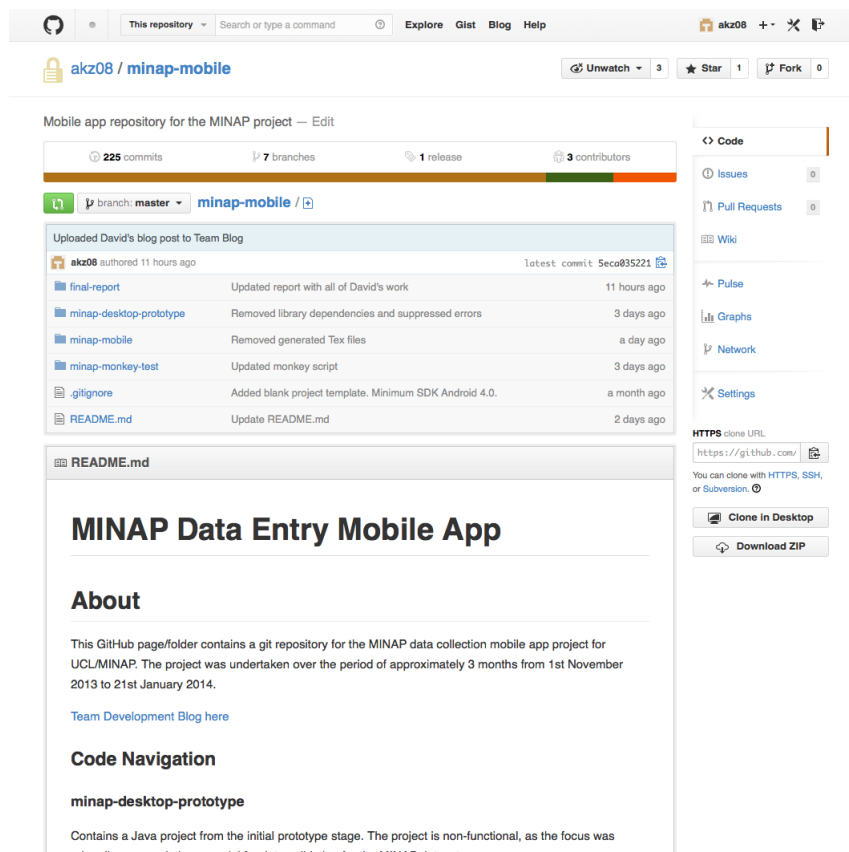

Figure 2: Snapshot of the team's GitHub repository page

Flowdock

For a unified cross-platform communication service, and to avoid having to check both Trello and GitHub frequently, we also used FlowDock extensively to easily track changes and keep each other updated. The service also allowed team members to control the level of notifications, to avoid a constant barrage of information (as well as work-life separation).

The service proved to be especially useful in tracking the progress of work packages, allowing other members to intervene if the pace of work was noticeably slower than expected.
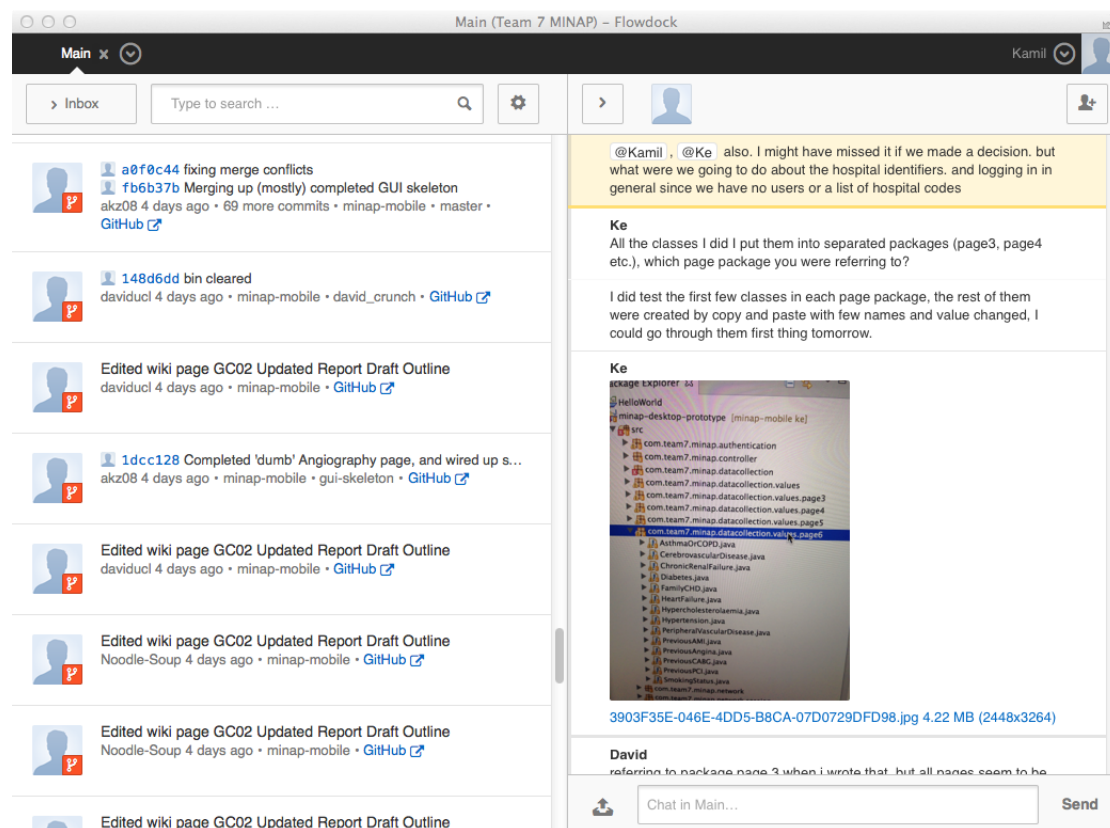
Figure 3: Snapshot of the team's Flowdock activity

11.2.2 Task Assignment

Team members typically were assigned work packages by the team leader, based on their current capabilities. These tended to be general work outlines, and were negotiable in their scope. The decisions on work packages at any given time were

strongly influenced by ongoing technology research, the current state of the overall system, and client feedback.

11.2.3 Meetings

The team typically met every workday to complete work together up to the first academic milestone. Most meetings since then have been conducted informally online using Flowdock, with occasional in-person catch-up meetings. This structure was more suited to the team once working on the final product, as work by each member was mostly independent of each other. Snapshots of early team meetings can be found in appendix C.

A   Data Dictionary

A.1   Patient Class

AdminStatus	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>adminStatus</u> : Byte	Holds the short code for patient's admission status
- <u>statusLongCode</u> : String	Holds the long code associated with adminStatus
+ <<constructor>> AdminStatus()	Sets the appropriate superclass values
+ <u>setAdminStatus</u>(admStatus : Byte) : Boolean	Returns true if parameter passes validation rules
+ <u>getAdminStatus</u>() : String	Returns the selected short code and corresponding long code as a String
+ <u>getLongCode</u>() : String	Returns the long code associated with adminStatus variable

AdmissionAfterNSTEMI	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>admissionAfterSTEMI</u> : Boolean = false	Holds a boolean value, reveals items on UI
+ <<constructor>> AdmissionAfterNSTEMI()	Sets the appropriate superclass values
+ <u>setAdmissionAfterSTEMI</u>(admASTEMI : Boolean) : Boolean	Sets, then returns admissionAfterSTEMI
+ <u>getAdmissionAfterSTEMI</u>() : String	Returns a String value of admissionAfterSTEMI

AdmissionDate	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import:>> com.ucl.appteam7.minapmobile.model.Value;	
<<import:>> java.text.ParseException;	
<<import:>> java.text.SimpleDateFormat;	
<<import:>> java.util.Date;	
- adminDate : String	Holds Date String, formatted as needed
- adminTime : String	Holds Time String, formatted as needed
- adminDateTime : String	Combines both previous variables
- now : Date = new Date()	Holds the current date, used for validation
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for adminDate
st : SimpleDateFormat = new SimpleDateFormat("HH:mm")	Holds the String format desired for adminTime
sdt : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyy HH:mm")	Holds the String format for adminDateTime
+ <<constructor>> AdmissionDate()	Sets the appropriate superclass values
+ setDateTime() : Boolean	Checks date and time validation, returns true if passed
+ setADate(aDate : Date) : Boolean	Checks the passed Date against date validation rules, returns true if passed
+ setATime(aTime : Date) : Boolean	Checks the passed Date against time validation rules, returns true if passed
+ getDateTime() : String	Returns adminDateTime variable
+ getADate() : String	Returns adminDate variable
+ getATime() : String	Returns adminTime variable

AdmissionMethod	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admissionMethod : Byte	Holds the short code for patient's admission method
- admissionLongCode : String	Holds the long code associated with admissionMethod
+ <<constructor>> AdmissionMethod()	Sets the appropriate superclass values
+ setAdmissionMethod(admMethod : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdmissionMethod() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with admissionMethod variable

AdmissionWard	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admissionWard : Byte	Holds the short code for patient's admission ward
- wardLongCode : String	Holds the long code associated with admissionWard
+ <<constructor>> AdmissionWard()	Sets the appropriate superclass values
+ setAdmissionWard(admWard : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdmissionWard() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with admissionWard variable

AdmittingConsultant	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admittingConsultant : Byte	Holds the short code for patient's admitting consultant
- consulLongCode : String	Holds the long code associated with admittingConsultant
+ <<constructor>> AdmittingConsultant()	Sets the appropriate superclass values
+ setAdmittingConsultant(admConsul : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdmittingConsultant() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with admittingConsultant variable

AngioDate	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for angioDate
st : SimpleDateFormat = new SimpleDateFormat("HH:mm")	Holds the String format desired for angioTime
sdt : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy HH:mm")	Holds the String format desired for angioDateTime
- angioDate : String	Holds the patient's angiography date
- angioTime : String	Holds the patient's angiography time
- angioDateTime : String	Holds the patient's angiography date and time
- *DATE_FORMAT* : String = "01/01/2000"	Lower bound for date range check
- now : Date = new Date()	Holds current Date, used for range check
+ <<constructor>> AngioDate()	Set the appropriate superclass values
+ setDateTime() : Boolean	Returns true if both dates passed validation
+ setangioralDate(rDate : Date) : Boolean	Returns true if Date parameter passes validation
+ setangioralTime(rTime : Date) : Boolean	Returns true if Time parameter passes validation
+ getDateTime() : String	Returns angioDateTime variable
+ getAngioDate() : String	Returns angioDate variable
+ getAngioTime() : String	Returns angioTime variable

AngioDelay	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- angioDelay : Byte	Holds the short code for patient's angiography delay
- angioDelayLongCode : String	Holds the long code associated with angioDelay
+ <<constructor>> AngioDelay()	Sets the appropriate superclass values
+ setAngioDelay(aDelay : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAngioDelay() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with angioDelay variable

AsthmaOrCOPD	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- asthmaCOPD : Byte	Holds the short code for patient's cerebrovascular disease status
- asthmaCOPDLongCode : String	Holds the long code associated with asthmaCOPD
+ <<constructor>> AsthmaOrCOPD()	Sets the appropriate superclass values
+ setAsthmaCOPD(cDisease : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAsthmaCOPD() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with asthmaCOPD variable

BMI	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>bmi</u> : Double	Holds the patient's BMI
+ <<constructor>> BMI()	Set the appropriate super-class values
+ <u>setBMI</u>(height : Double, weight : Double) : Boolean	Calculates BMI, returns true if parameter passes validation
+ <u>getBMI</u>() : String	Returns bmi as a formatted String

CerebrovascularDisease	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>cerebrovascularDisease</u> : Byte	Holds the short code for patient's cerebrovascular disease status
- <u>cerebrovascularDiseaseLongCode</u> : String	Holds the long code associated with cerebrovascularDisease
+ <<constructor>> CerebrovascularDisease()	Sets the appropriate super-class values
+ <u>setCerebrovascular</u>(cDisease : Byte) : Boolean	Returns true if parameter passes validation rules
+ <u>getCerebrovascular</u>() : String	Returns the selected short code and corresponding long code as a String
+ <u>getLongCode</u>() : String	Returns the long code associated with cerebrovascularDisease variable

ChronicRenalFailure	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- chronicRenalFailure : Byte	Holds the short code for patient's chronic renal failure status
- chronicRenalFailureLongCode : String	Holds the long code associated with chronicRenalFailure
+ <<constructor>> ChronicRenalFailure()	Sets the appropriate superclass values
+ setRenalFailure(cRenalFailure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getRenalFailure() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with chronicRenalFailure variable

CoronaryAngiography	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- coronaryAngiography : Byte	Holds the short code for patient's coronary angiography
- coronaryAngiographyLongCode : String	Holds the long code associated with coronaryAngiography
+ <<constructor>> CoronaryAngiography()	Sets the appropriate superclass values
+ setCoronaryAngiography(cAngiography : Byte) : Boolean	Returns true if parameter passes validation rules
+ getCoronaryAngiography() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with coronaryAngiography variable

CoronaryIntervention	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- coronaryIntervention : Byte	Holds the short code for patient's coronary intervention
- coronaryInterventionLongCode : String	Holds the long code associated with coronaryIntervention
+ <<constructor>> CoronaryIntervention()	Sets the appropriate superclass values
+ setCoronaryIntervention(cIntervention : Byte) : Boolean	Returns true if parameter passes validation rules
+ getCoronaryIntervention() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with coronaryIntervention variable

DaycaseTransferDate	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
<u>sd</u> : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for daycaseDate
- <u>daycaseDate</u> : String	Holds the patient's daycase transfer date
- <u>*DATE_FORMAT*</u> : String = "01/01/2000"	Lower bound for date range check
- <u>now</u> : String = sd.format(new Date())	Holds current Date as a String, used for range check
+ <<constructor>> DaycaseTransferDate()	Set the appropriate super-class values
+ <u>setDaycaseDate</u>(iDate: Date) : Boolean	Returns true if parameter passes validation
+ <u>getDaycaseDate</u>() : String	Returns daycaseDate variable

DBAdapter	Holds all SQLite logic
~com.ucl.appteam7.minapmobile.model	
<<import>> java.util.Collections	
<<import>> java.util.Date	
<<import>> java.util.List	
<<import>> java.util.ArrayList	
<<import>> android.content.ContentValues	
<<import>> android.content.Context	
<<import>> android.database.Cursor	
<<import>> android.database.SQLException	
<<import>> android.database.sqlite.SQLiteOpenHelper	
<<import>> android.database.sqlite.SQLiteDatabase	
<<import>> android.util.Log	
+ $HOSPITAL\_IDENTIFIER$: String = "HospitalID"	Declares HospitalID as a column for a database's table
+ $RECORD\_NO$: String = "RecordNum"	Declares RecordNum
+ NHS_NUMBER : String = "NHSNumber"	Declares NHSNumber
+ $PATIENT\_SURNAME$: String = "PatientSurname"	Declares PatientSurname
+ $PATIENT\_FORENAME$: String = "PatientForename"	Declares PatientForename
+ $PATIENT\_DOB$: String = "PatientDOB"	Declares PatientDOB
+ $ADMISSION\_DATE$: String = "AdminDate"	Declares AdminDate
+ $PATIENT\_INFO$: String[] = $\{HOSPITAL\_IDENTIFIER, RECORD\_NO,$ $NHS\_NUMBER, PATIENT\_SURNAME,$ $PATIENT\_FORENAME, PATIENT\_DOB,$ $ADMISSION\_DATE\}$	Compiles previous columns for the Patient Info page
+ $INITIAL\_DIAGNOSIS$: String = "InitialDiagosis"	Declares InitialDiagnosis
+ $ADMISSION\_AFTER\_NSTEMI$: String = "AdminStemi"	Declares AdminStemi
+ $HIGH\_RISK\_NSTEMI$: String = "HiRisknSTEMI"	Declares HiRisknSTEMI
+ $INTERVENTIONAL\_PROCEDURE$: String = "Interventional-Procedure"	Declares InterventionalProcedure
+ $RETURN\_TO\_REFERRING\_HOSPITAL$: String = "ReferHospitalReturn"	Declares ReferHospiralReturn
+ $INTERVENTIONAL\_CENTRE\_CODE\_ID$: String = "InterventionCentreCode"	Declares InterventionCentreCode

+ _DIAGNOSIS_ : String[] = {_INITIAL_DIAGNOSIS_, _ADMISSION_AFTER_NSTEMI_, _HIGH_RISK_NSTEMI_, _INTERVENTIONAL_PROCEDURE_, _RETURN_TO_REFERRING_HOSPITAL_, _INTERVENTIONAL_CENTRE_CODE_ID_}	Compiles previous columns for Initial Diagnosis page
+ _PATIENT_GENDER_ : String = "Gender"	Declares Gender
+ _PATIENT_ETHNICITY_ : String = "Ethnicity"	Declares Ethnicity
+ _ADMISSION_METHOD_ : String = "AdminMethod"	Declares AdminMethod
+ _ADMISSION_WARD_ : String = "AdminWard"	Declares AdminWard
+ _GP_PCT_CODE_ : String = "GPCode"	Declares GPCode
+ _PATIENT_POST_CODE_ : String = "PostCode"	Declares PostCode
+ _ADMITTING_CONSULTANT_ : String = "AdminConsul"	Declares AdminConsul
+ _ADMISSION_STATUS_ : String = "AdminStatus"	Declares AdminStatus
+ _PLACE_FIRST_12_LEAD_ECG_PERFORMED_ : String = "FirstECG"	Declares FirstECG
+ _NHS_VERIFICATION_ : String = "NHSVerif"	Declares NHSVerif
+ _REFERRAL_HOSPITAL_ : String = "RefHospital"	Declares RefHospital
+ _ADMISSION_ : String[] = {_PATIENT_GENDER_, _PATIENT_ETHNICITY_, ADMISSION_METHOD_, _ADMISSION_WARD_, GP_PCT_CODE_, PATIENT_POST_CODE_, _ADMITTING_CONSULTANT_, ADMISSION_STATUS_, _PLACE_FIRST_12_LEAD_ECG_PERFORMED_, _NHS_VERIFICATION_, REFERRAL_HOSPITAL_}	Compiles previous columns for Demographics and Admission page
+ _INITIAL_REPERFUSION_TREATMENT_ : String = "InitialReperfusionTreat"	Declares InitialReperfusionTreat
+ _REPERFUSION_NOT_GIVEN_ : String = "ReperNotGiven"	Declares ReperNotGiven
+ _ECG_DETERMINING_TREATMENT_ : String = "ECGDetermineTreat"	Declares ECGDetermineTreat
+ _ECG_QRS_COMPLEX_DURATION_ : String = "_ECG_QRSComplex_"	Declares _ECG_QRSComplex_
+ _STEMI_LOCATION_ : String = "LocationSTEMI"	Declares LocationSTEMI
+ _INTERVENTIONAL_CENTRE_CODE_ : String = "ReperInterventionCentre"	Declares ReperInterventionCentre
+ _INFARCTION_SITE_ : String = "InfarctionSite"	Declares InfarctionSite
+ _REPERFUSION_ : String[] = {_INITIAL_REPERFUSION_TREATMENT_, _REPERFUSION_NOT_GIVEN_, _ECG_DETERMINING_TREATMENT_, _ECG_QRS_COMPLEX_DURATION_, STEMI_LOCATION_, _INTERVENTIONAL_CENTRE_CODE_, INFARCTION_SITE_}	Compiles previous columns for Initial Reperfusion page

36

+ _CORONARY_ANGIO_ : String = "CoronaryAngio"	Declares CoronaryAngio
+ _REFERRAL_DATE_ : String = "ReferralDate"	Declares ReferralDate
+ _ANGIO_PERFORM_DELAY_ : String = "AngioDelay"	Declares AngioDelay
+ _ANGIO_DATE_ : String = "AngioDate"	Declares AngioDate
+ _INTERVENTIONAL_CENTRE_CODE_AN_ : String = "AngioInterventionCentre"	Declares AngioInterventionCentre
+ _LOCAL_INTERVENTION_ : String = "LocalIntervention"	Declares LocalIntervention
+ _CORONARY_INTERVENTION_ : String = "CoronaryIntervention"	Declares CoronaryIntervention
+ _PATIENT_RETURN_ : String = "ReturnExpected"	Declares ReturnExpected
+ _DAYCASE_TRANSFER_ : String = "DaycaseTransfer"	Declares DaycaseTransfer
+ _REFERRING_RETURN_ : String = "ReferringHospitalReturn"	Declares ReferringHospitalReturn
+ _ANGIOGRAPHY_ : String[] = {_CORONARY_ANGIO_, _REFERRAL_DATE_, ANGIO_PERFORM_DELAY_, ANGIO_DATE_, INTERVENTIONAL_CENTRE_CODE_AN_, LOCAL_INTERVENTION_, CORONARY_INTERVENTION_, PATIENT_RETURN_, DAYCASE_TRANSFER_, REFERRING_RETURN_}	Compiles previous columns for Angiography page
+ _SYSTOLIC_ : String = "SystolicBP"	Declares SystolicBP
+ _HEART_RATE_ : String = "HeartRate"	Declares HeartRate
+ _KILLIP_CLASS_ : String = "KillipClass"	Declares KillipClass
+ _BMI_ : String = "BMI"	Declares BMI
+ _HEIGHT_ : String = "Height"	Declares Height
+ _WEIGHT_ : String = "Weight"	Declares Weight
+ _EXAMINATIONS_ : String[] = {_SYSTOLIC_, HEART_RATE_, KILLIP_CLASS_, BMI_, HEIGHT_, WEIGHT_}	Compiles previous columns for Examinations page
+ _PREVIOUS_AMI_ : String = "PrevAMI"	Declares PrevAMI
+ _HYPERTENSION_ : String = "Hypertension"	Declares Hypertension
+ _CEREBROVASCULAR_ : String = "CerebroDisease"	Declares CerebroDisease
+ _PREVIOUS_PCI_ : String = "PrevPCI"	Declares PrevPCI
+ _SMOKING_ : String = "Smoking"	Declares Smoking
+ _DIABETES_ : String = "Diabetes"	Declares Diabetes
+ _PREVIOUS_ANGINA_ : String = "PrevAngina"	Declares PrevAngina
+ _HYPERCHOLESTEROL_ : String = "Hypercholesterol"	Declares Hypercholesterol
+ _ASTHMA_COPD_ : String = "AsthmaCOPD"	Declares AsthmaCOPD
+ _PREVIOUS_CABG_ : String = "PrevCABG"	Declares PrevCABG
+ _HEART_FAILURE_ : String = "HeartFailure"	Declares HeartFailure
+ _PV_DISEASE_ : String = "PeriphVascularDisease"	Declares PeriphVascularDisease

+ _RENAL_FAILURE_ : String = "RenalFailure"	Declares RenalFailure
+ _FAMILY_CHD_ : String = "FamilyCHD"	Declares FamilyCHD
+ _MEDICAL_HISTORY_ : String[] = {$PREVIOUS\_AMI$, $HYPERTENSION$, $CEREBROVASCULAR$, $PREVIOUS\_PCI$, $SMOKING$, $DIABETES$, $PREVIOUS\_ANGINA$, $HYPERCHOLESTEROL$, $ASTHMA\_COPD$, $PREVIOUS\_CABG$, $HEART\_FAILURE$, $PV\_DISEASE$, $RENAL\_FAILURE$, $FAMILY\_CHD$}	Compiles previous columns for Medical History page
+ _TAG_ : String = "DBAdapter"	Used when updating the Database's schema
- _DATABASE_NAME_ : String = "minap"	Names the Database
- _TABLE_NAME_ : String = "patient"	Names the patient table
- _DATABASE_VERSION_ : Integer = 11	Must be increase when the schema is changed
- _DATABASE_CREATE_ : String	Holds the SQL Statement that creates the Database and table
- Context : Context	Provides context for the session when needed
- DBHelper : DatabaseHelper	Used for helping SQLite operations
- db : SQLiteDatabase	The Database itself
- <<class>> DatabaseHelper()	¡¡extends SQLiteOpen-Helper¿¿ Inner class, creates and/or updates the database
+ <<override>> onCreate(db : SQLiteDatabase) : void	Creates the database, throws exception if unable to
+ <<override>> onUpgrade(db : SQLiteDatabase, oldVersion : Integer, newVersion : Integer) : void	Clears, then upgrades the database
- concatenateArrays(page0 : String[], page1 : String[], page2 : String[], page3 : String[], page4 : String[], page5 : String[], page6 : String[]) : String[]	Concatenates all page column arrays
+ <<constructor>> DBAdapter(ctx : Context)	Uses DBHelper to create and operate the database
+ open() : DBAdapter	Returns the writable database
+ close() : void	Closes the database

+ insertPatient(id : String, forename : String, surname : String, dob : String, nhs : String, hId : String, admDate : String) : Boolean	Returns true if parameters have been inserted onto database's patient info section
+ updatePatientInfo(oldid : String, newid : String, forename : String, surname : String, dob : String, nhs : String, hId : String, admDate : String) : Boolean	Returns true if parameters were used to update database's patient info section
+ getPatientInfo(id : String) : Cursor	Returns a cursor object containing the database's patient info section where id = recordNumber
+ updateInitialDiagnosis(id : String, diagnosis : String, admission : String, nstemi : String, procedure : String, refer : String, code : String) : Boolean	Returns true if parameters were used to update the database's initial diagnosis section
+ getDiagnosis(id : String) : Cursor	Returns a cursor object containing the database's initial diagnosis section where id = recordNumber
+ updateDemographics(id : String, gender : String, ethnicity : String, method : String, ward : String, gpcode : String, post : String, consultant : String, status : String, ecg, verif : String, refer : String) : Boolean	Returns true if parameters were used to update the database's Demographics and Admission section
+ getDemographics(id : String) : Cursor	Returns a cursor object containing the database's demographics and admission section where id = recordNumber
+ updateInitialReperfusion(id : String, treatment : String, reperfusion : String, ecgtreat : String, ecgqrs : String, location : String, iccode : String, infarction : String) : Boolean	Returns true if parameters were used to update the database's Initial Reperfusion section
+ getReperfusion(id : String) : Cursor	Returns a cursor object containing the database's initial reperfusion section where id = recordNumber
+ updateAngiography(id : String, angio : String, drefer : String, adelay : String, adate : String, acode : String, inter : String, coronary : String, patret : String, daycase : String, refret : String) : Boolean	Returns true if parameters were used to update the database's Angiography section

+ getAngiography(id : String) : Cursor	Returns a cursor object containing the database's angiography section where id = recordNumber
+ updateExaminations(id : String, systolic : String, heart : String, killip : String, bmi : String, height : String, weight : String) : Boolean	Returns true if parameters were used to update the database's Examinations section
+ getExaminations(id : String) : Cursor	Returns a cursor object containing the database's examinations section where id = recordNumber
+ updateMedicalHistory(id : String, ami : String, tension : String, cerebro : String, pci : String, smoke : String, diabetes : String, angina : String, choles : String, asthma : String, cabg : String, heart : String, vascular : String, renal : String, chd : String) : Boolean	Returns true if parameters were used to update the database's Medical History section
+ getMedicalHistory(id : String) : Cursor	Returns a cursor object containing the database's medical history section where id = recordNumber
+ deletePatient(id : String) : Boolean	Returns true if record where id = recordNum has been deleted
+ allPatients() : Cursor	Returns a cursor object containing all information on the database

Diabetes	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- diabetes : Byte	Holds the short code for patient's diabetes status
- diabetesLongCode : String	Holds the long code associated with diabetes
+ <<constructor>> Diabetes()	Sets the appropriate superclass values
+ setDiabetes(diabete : Byte) : Boolean	Returns true if parameter passes validation rules
+ getDiabetes() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with diabetes variable

DOB	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for dob
- dob : String	Holds the patient's date of birth
- $DATE\_FORMAT$: String = "01/01/1900"	Lower bound for date range check
- twentyYrsAgo : String = "01/01/1994"	Upper bound for date range check
+ <<constructor>> DOB()	Set the appropriate superclass values
+ setDOB(bDate : Date) : Boolean	Checks the passed Date against validation rules, returns true if passed
+ getDOB() : String	Returns dob variable

EcgDetermineTreatment	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- ecgDetermineTreatment : Byte	Holds the short code for patient's ECG determining treatment
- ecgDetermineTreatmentLongCode : String	Holds the long code associated with ecgDetermine-Treatment
+ <<constructor>> EcgDetermineTreatment()	Sets the appropriate super-class values
+ setECGTreatment(eDetermineTreatment : Byte) : Boolean	Returns true if parameter passes validation rules
+ getECGTreatment() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ecgDetermine-Treatment variable

EcgQRSComplex	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- ecgQRSComplex : Byte	Holds the short code for patient's ECG / QRS Complex
- ecgQRSComplexLongCode : String	Holds the long code associated with ecgQRSComplex
+ <<constructor>> EcgQRSComplex()	Sets the appropriate super-class values
+ setECGQRSComplex(eQRSComplex : Byte) : Boolean	Returns true if parameter passes validation rules
+ getECGQRSComplex() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ecgQRSComplex variable

FamilyCHD	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- familyCHD : Byte	Holds the short code for patient's family CHD status
- familyCHDLongCode : String	Holds the long code associated with familyCHD
+ <<constructor>> FamilyCHD()	Sets the appropriate superclass values
+ setFamilyCHD(hFailure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getFamilyCHD() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with familyCHD variable

FirstECG	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- firstECG : Byte	Holds the short code for patient's first ECG instance
- ecgLongCode : String	Holds the long code associated with firstECG
+ <<constructor>> FirstECG()	Sets the appropriate superclass values
+ setFirstECG(fECG : Byte) : Boolean	Returns true if parameter passes validation rules
+ getFirstECG() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with firstECG variable

Gender	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- patientGender : Byte	Holds the short code for patient's gender
- genderLongCode : String	Holds the long code associated with patientGender
+ <<constructor>> Gender()	Sets the appropriate super-class values
+ setPatientGender(gen : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPatientGender() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with patientGender variable

GPCode	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- gpCode : String	Holds the patient's GP code
+ VAL_LENGTH : Short = 6	Holds the maximum length for gpCode
+ <<constructor>> GPCode()	Sets the appropriate super-class values
+ setGPCode(pctCode : String) : Boolean	Returns true if parameter passes validation rules
+ getGPCode() : String	Returns gpCode

HeartFailure	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- heartFailure : Byte	Holds the short code for patient's heart failure status
- heartFailureLongCode : String	Holds the long code associated with heartFailure
+ <<constructor>> HeartFailure()	Sets the appropriate superclass values
+ setHeartFailure(hFailure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHeartFailure() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with heartFailure variable

HeartRate	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- heartRate : Double	Holds the patient's heart rate
+ <<constructor>> HeartRate()	Set the appropriate superclass values
+ setHeartRate(hRate : Double) : Boolean	Returns true if parameter passes validation
+ getHeartRate() : String	Returns heartRate

Height	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- height : Double	Holds the patient's height
+ <<constructor>> Height()	Set the appropriate super-class values
+ setHeight(h : Double) : Boolean	Returns true if parameter passes validation
+ getHeight() : String	Returns height

HiRisknSTEMI	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hiRisknSTEMI : Byte	Holds the short code choice for High Risk nSTEMI
- nSTEMILongCode : String	Holds the long code associated with hiRisknSTEMI
+ <<constructor>> HiRisknSTEMI()	Sets the appropriate super-class values
+ setHiRisknSTEMI(hiNSTEMI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHiRisknSTEMI() : String	Returns the selected short code and corresponding long code as a String

HospitalIdentifier	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hospitalID : String	Holds a three letter string for hospitalID
+ VAL_LENGTH : Short = 3	Holds the valid length for hospitalID
+ <<constructor>> HospitalIdentifier()	Sets the appropriate super-class values
+ setHospitalIdentifier(hospId : String) : Boolean	Checks the paramenter against validation rules, returns true if passed
+ getHospitalIdentifier() : String	Returns hospitalID

Hypercholesterolaemia	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hyperCholesterolaemia : Byte	Holds the short code for patient's hypercholesterolaemia status
- hyperCholesterolaemiaLongCode : String	Holds the long code associated with hyperCholesterolaemia
+ <<constructor>> Hypercholesterolaemia()	Sets the appropriate super-class values
+ setHypercholesterol(hCholesterolaemia : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHypercholesterol() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with hyperCholesterolaemia variable

Hypertension	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hyperTension : Byte	Holds the short code for patient's hypertension status
- hyperTensionLongCode : String	Holds the long code associated with hyperTension
+ <<constructor>> Hypertension()	Sets the appropriate superclass values
+ setHypertension(hTension : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHypertension() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with hyperTension variable

InitialDiagnosis	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- initialDiagnosis : Byte	Holds the short code choice for Initial Diagnosis
- diagnosisLongCode : String	Holds the long code associated with initialDiagnosis
+ <<constructor>> InitialDiagnosis()	Sets the appropriate superclass values
+ setInitialDiagnosis(iniDiagnosis : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInitialDiagnosis() : String	Returns the selected short code and corresponding long code as a String
+ getInitialLongCode() : String	Returns diagnosisLongCode

InitialReperfusion	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- initialReperfusion : Byte	Holds the short code for patient's initial reperfusion
- initialReperfusionLongCode : String	Holds the long code associated with initialReperfusion
+ <<constructor>> InitialReperfusion()	Sets the appropriate superclass values
+ setInitialReperfusion(iReperfusion : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInitialReperfusion() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with initialReperfusion variable

InfarctionSite	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- infarctionSite : Byte	Holds the short code for patient's infarction site
- infarctionSiteLongCode : String	Holds the long code associated with infarctionSite
+ <<constructor>> InfarctionSite()	Sets the appropriate superclass values
+ setInfarctionSite(iSite : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInfarctionSite() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with infarctionSite variable

InterventionalCentreCode	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- interventionalID : String	Holds the patient's Interventional Centre Code
+ VAL_LENGTH : Short = 3	Holds the maximum length for interventionalID
+ <<constructor>> InterventionalCentreCode()	Sets the appropriate superclass values
+ setInterventionalCentre(iCode : String) : Boolean	Returns true if parameter passes validation rules
+ getInterventionalCentre() : String	Returns interventionalID
Note: This class is used three times within the Patient singleton	

InterventionalProcedure	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- interventionalProcedure : Byte	Holds the short code for Interventional Procedure
- procedureLongCode : String	Holds the long code associated with interventionalProcedure
+ <<constructor>> InterventionalProcedure()	Sets the appropriate superclass values
+ setInterventionalProcedure(interHosProcedure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInterventionalProcedure() : String	Returns the selected short code and corresponding long code as a String

KillipClass	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- killipClass : Byte	Holds the short code for patient's coronary angiography
- killipClassLongCode : String	Holds the long code associated with killipClass
+ <<constructor>> KillipClass()	Sets the appropriate superclass values
+ setKillipClass(kClass : Byte) : Boolean	Returns true if parameter passes validation rules
+ getKillipClass() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with killipClass variable

LocationAtSTEMI	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- locationAtSTEMI : Byte	Holds the short code for patient's location at STEMI
- locationAtSTEMILongCode : String	Holds the long code associated with locationAtSTEMI
+ <<constructor>> LocationAtSTEMI()	Sets the appropriate superclass values
+ setSTEMILocation(lAtSTEMI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getSTEMILocation() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with locationAtSTEMI variable

LocalInterventionDate	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
<u>sd</u> : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for interventionDate
- <u>interventionDate</u> : String	Holds the patient's local intervention date
- <u>*DATE_FORMAT*</u> : String = "01/01/2000"	Lower bound for date range check
- <u>now</u> : String = sd.format(new Date())	Holds current Date as a String, used for range check
+ <<constructor>> LocalInterventionDate()	Set the appropriate super-class values
+ <u>setInterventionDate</u>(iDate: Date) : Boolean	Returns true if parameter passes validation
+ <u>getInterventionDate</u>() : String	Returns interventionDate variable

NHSNumber	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>nhsNumber</u> : String	Holds the patient's NHS number
+ <u>*VAL_LENGTH*</u> : Short = 10	Holds the maximum length for nhsNumber
+ <<constructor>> NHSNumber()	Sets the appropriate super-class values
+ <<setNHSNum>>(nhsNum : String) : Boolean	Checks the parameter against validation rules, returns true if passed
+ <<getNHSNum>>() : String	Returns nhsNumber

NHSVerification	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- verification : Byte	Holds the short code for patient's NHS verification
- verificationLongCode : String	Holds the long code associated with verification
+ <<constructor>> NHSVerification()	Sets the appropriate superclass values
+ setVerification(verif : Byte) : Boolean	Returns true if parameter passes validation rules
+ getVerification() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with verification variable

Patient	<<singleton>>
˜com.ucl.appteam7.minapmobile.model	
<<import>> com.ucl.appteam7.minapmobile.model.values.PatientInfo.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.InitialReperfusion.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.Angiography.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.Examinations.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.MedicalHistory.*	
- instance : Patient	Controls the singleton behavior of this class
+ HospitalIdentifier : HospitalIdentifier	Declares a HospitalIdentifier object to the singleton
+ RecordNumber : PatientCaseRecordNumber	Declares a PatientCaseRecordNumber object to the singleton
+ NHSNumber : NHSNumber	Declares a NHSNumber object to the singleton
+ Surname : PatientSurname	Declares a PatientSurname object to the singleton
+ Forename : PatientForename	Declares a PatientForename object to the singleton
+ DOB : DOB	Declares a DOB object to the singleton
+ AdmissionDate : AdmissionDate	Declares a AdmissionDate object to the singleton
+ InitialDiagnosis : InitialDiagnosis	Declares a InitialDiagnosis object to the singleton
+ AdmissionAfterNSTEMI : AdmissionAfterNSTEMI	Declares a AdmissionAfterNSTEMI object to the singleton
+ HighRisknSTEMI : HiRisknSTEMI	Declares a HiRisknSTEMI object to the singleton

+ InterventionalProcedure : InterventionalProcedure	Declares a Interventional-Procedure object to the singleton
+ ReferHospitalReturn : ReferHospitalReturn	Declares a ReferHospitalReturn object to the singleton
+ InterventionalCentreCode : InterventionalCentreCode	Declares a Interventional-CentreCode object to the singleton
+ Gender : Gender	Declares a Gender object to the singleton
+ Ethnicity : PatientEthnicity	Declares a PatientEthnicity object to the singleton
+ AdmissionMethod : AdmissionMethod	Declares a Admission-Method object to the singleton
+ AdmissionWard : AdmissionWard	Declares a AdmissionWard object to the singleton
+ GPCode : GPCode	Declares a GPCode object to the singleton
+ PostCode : PatientPostcode	Declares a PatientPostcode object to the singleton
+ AdmitConsul : AdmittingConsultant	Declares a AdmittingConsultant object to the singleton
+ AdminStatus : AdminStatus	Declares a AdminStatus object to the singleton
+ FirstECG : FirstECG	Declares a FirstECG object to the singleton
+ NHSVerification : NHSVerification	Declares a NHSVerification object to the singleton
+ ReferralHospital : ReferralHospital	Declares a ReferralHospital object to the singleton
+ InitialReperfusion : InitialReperfusion	Declares a InitialReperfusion object to the singleton
+ ReperfusionNotGiven : ReperfusionNotGiven	Declares a ReperfusionNot-Given object to the singleton
+ ECGTreatment : EcgDetermineTreatment	Declares a EcgDetermine-Treatment object to the singleton

+ *ECG_QRSComplex* : EcgQRSComplex	Declares a EcgQRSComplex object to the singleton
+ LocationSTEMI : LocationAtSTEMI	Declares a LocationAt-STEMI object to the singleton
+ ReperfusionCentreCode : InterventionalCentreCode	Declares a second InterventionalCentreCode object to the singleton
+ InfarctionSite : InfarctionSite	Declares a InfarctionSite object to the singleton
+ CoronaryAngiography : CoronaryAngiography	Declares a CoronaryAngiography object to the singleton
+ ReferralDate : ReferralDate	Declares a ReferralDate object to the singleton
+ AngioDelay : AngioDelay	Declares a AngioDelay object to the singleton
+ AngioDate : AngioDate	Declares a AngioDate object to the singleton
+ AngioCentreCode : InterventionalCentreCode	Declares a third InterventionalCentreCode object to the singleton
+ InterventionDate : LocalInterventionDate	Declares a LocalInterventionDate object to the singleton
+ CoronaryIntervention : CoronaryIntervention	Declares a CoronaryIntervention object to the singleton
+ ReturnExpected : ReturnExpected	Declares a ReturnExpected object to the singleton
+ DaycaseTransfer : DaycaseTransferDate	Declares a DaycaseTransferDate object to the singleton
+ ReferReturnDate : ReferHospitalReturnDate	Declares a ReferHospitalReturnDate object to the singleton
+ Systolic : SystolicBP	Declares a SystolicBP object to the singleton
+ HeartRate : HeartRate	Declares a HeartRate object to the singleton

+ Killip : KillipClass	Declares a KillipClass object to the singleton
+ Height : Height	Declares a Height object to the singleton
+ Weight : Weight	Declares a Weight object to the singleton
+ BMI : BMI	Declares a BMI object to the singleton
+ PreviousAMI : PreviousAMI	Declares a PreviousAMI object to the singleton
+ Hypertension : Hypertension	Declares a Hypertension object to the singleton
+ Cerebrovascular : CerebrovascularDisease	Declares a CerebrovascularDisease object to the singleton
+ PreviousPCI : PreviousPCI	Declares a PreviousPCI object to the singleton
+ Smoking : SmokingStatus	Declares a SmokingStatus object to the singleton
+ Diabetes : Diabetes	Declares a Diabetes object to the singleton
+ PreviousAngina : PreviousAngina	Declares a PreviousAngina object to the singleton
+ HyperCholesterol : Hypercholesterolaemia	Declares a Hypercholesterolaemia object to the singleton
+ AsthmaCOPD : AsthmaOrCOPD	Declares a AsthmaOrCOPD object to the singleton
+ PreviousCABG : PreviousCABG	Declares a PreviousCABG object to the singleton
+ HeartFailure : HeartFailure	Declares a HeartFailure object to the singleton
+ PeripheralVascular : PeripheralVascularDisease	Declares a PreipheralVascularDisease object to the singleton
+ RenalFailure : ChronicRenalFailure	Declares a ChronicRenalFailure object to the singleton

+ FamilyCHD : FamilyCHD	Declares a FamilyCHD object to the singleton
+ <u>get</u>() : Patient	Returns the instance variable, or creates one if it does not exist
- <<constructor>> Patient()	Private constructor, restricts creation of Patient objects to this class only
# <<override>> clone() : Object	Forbids a clone() method call. Completes the singleton declaration

PatientCaseRecordNumber	**<<extends Value>>**
~com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>caseNumber</u> : String	Hold the patient's Case Number
- <u>oldCase</u> : String = caseNumber	Holds a copy of a patient's previous record number, used when updating Database
- VAL_LENGTH : Short = 10	Holds the maximum length for caseNumber
+ <<constructor>> PatientCaseRecordNumber()	Sets the appropriate super-class values
+ <<setCaseNumber>>(cNumber : String) : Boolean	Returns true if parameter passes validation rules
+ <<getCaseNumber>>()	Returns caseNumber
+ <<getOldCase>>()	Returns oldCase

PatientEthnicity	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- ethnicity : Byte	Holds the short code for patient's Ethnicity
- ethnicLongCode : String	Holds the long code associated with ethnicity
+ <<constructor>> PatientEthnicity()	Sets the appropriate superclass values
+ setEthnicity(ethnic : Byte) : Boolean	Returns true if parameter passes validation rules
+ getEthnicity() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ethnicity variable

PatientForename	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- forename : String	Holds the patient's forename
+ VAL_LENGTH : Short = 35	Holds the maximum length for forename
+ <<constructor>> PatientForename()	Sets the appropriate superclass values
+ setForename(fName : String) : Boolean	Returns true if parameter passes validation rules
+ getForename() : String	Returns forename variable

PatientPostcode	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- postCode : String	Holds the patient's Postcode
+ VAL_LENGTH : Short = 8	Holds the maximum length for postCode
+ <<constructor>> PatientPostcode()	Sets the appropriate superclass values
+ setPostcode(rId : String) : Boolean	Returns true if parameter passes validation rules
+ getPostcode() : String	Returns postCode

PatientSurname	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- surname : String	Holds the patient's surname
+ VAL_LENGTH : Short = 35	Holds the maximum length for surname
+ <<constructor>> PatientSurname()	Sets the appropriate superclass values
+ setSurname(sName : String) : Boolean	Returns true if parameter passes validation rules
+ getSurname() : String	Returns surname variable

PeripheralVascularDisease	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- peripheralVascularDisease : Byte	Holds the short code for patient's peripheral vascular disease
- peripheralVascularDiseaseLongCode : String	Holds the long code associated with peripheralVascularDisease
+ <<constructor>> PeripheralVascularDisease()	Sets the appropriate superclass values
+ setPeripheralVascularDisease(pVascularDisease : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPeripheralVascularDisease() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with peripheralVascularDisease variable

PreviousAMI	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousAMI : Byte	Holds the short code for patient's previous AMI status
- previousAMILongCode : String	Holds the long code associated with previousAMI
+ <<constructor>> PreviousAMI()	Sets the appropriate superclass values
+ setPreviousAMI(pAMI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousAMI() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousAMI variable

PreviousAngina	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousAngina : Byte	Holds the short code for patient's previous angina status
- previousAnginaLongCode : String	Holds the long code associated with previousAngina
+ <<constructor>> PreviousAngina()	Sets the appropriate superclass values
+ setPreviousAngina(pAngina : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousAngina() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousAngina variable

PreviousCABG	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousCABG : Byte	Holds the short code for patient's previous CABG status
- previousCABGLongCode : String	Holds the long code associated with previousCABG
+ <<constructor>> PreviousCABG()	Sets the appropriate superclass values
+ setPreviousCABG(pCABG : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousCABG() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousCABG variable

PreviousPCI	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousPCI : Byte	Holds the short code for patient's previous PCI status
- previousPCILongCode : String	Holds the long code associated with previousPCI
+ <<constructor>> PreviousPCI()	Sets the appropriate superclass values
+ setPreviousPCI(pPCI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousPCI() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousPCI variable

ReferHospitalReturn	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
- rDate : String	Holds the Date for return from Referring Hospital
- now : Date = new Date()	Holds the current date used for validation
sd SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the string format desired for rDate
+ <<constructor>> ReferHospitalReturn()	Sets the appropriate superclass values
+ setRDate(bDate : Date) : Boolean	Returns true if paramenter passes validation rules
+ getRDate() : String	Returns rDate

ReferHospitalReturnDate	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for returnDate
- returnDate : String	Holds the patient's date of return to referral hospital
- *DATE_FORMAT* : String = "01/01/2000"	Lower bound for date range check
- now : String = sd.format(new Date())	Holds current Date as a String, used for range check
+ <<constructor>> ReferHospitalReturnDate()	Set the appropriate superclass values
+ setReturnDate(rhrDate: Date) : Boolean	Returns true if parameter passes validation
+ getReturnDate() : String	Returns returnDate variable

ReferralDate	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for referDate
st : SimpleDateFormat = new SimpleDateFormat("HH:mm")	Holds the String format desired for referTime
sdt : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy HH:mm")	Holds the String format desired for referDateTime
- referDate : String	Holds the patient's referral date
- *DATE_FORMAT* : String = "01/01/2000"	Lower bound for date range check
- now : Date = new Date()	Holds current Date, used for range check
+ <<constructor>> ReferralDate()	Set the appropriate superclass values
+ setDateTime() : Boolean	Returns true if both dates passed validation
+ setReferralDate(rDate : Date) : Boolean	Returns true if Date parameter passes validation
+ setReferralTime(rTime : Date) : Boolean	Returns true if Time parameter passes validation
+ getDateTime() : String	Returns referDateTime variable
+ getReferralDate() : String	Returns referDate variable
+ getReferralTime() : String	Returns referTime variable

ReferralHospital	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- referralId : String	Holds the patient's ReferralHospital
+ VAL_LENGTH : Short = 3	Holds the maximum length for referralId
+ <<constructor>> ReferralHospital()	Sets the appropriate superclass values
+ setReferralIdentifier(rId : String) : Boolean	Returns true if parameter passes validation rules
+ getReferralIdentifier() : String	Returns referralID

ReperfusionNotGiven	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- reperfusionNotGiven : Byte	Holds the short code for reason reperfusion was not given
- rNotGivenLongCode : String	Holds the long code associated with reperfusionNotGiven
+ <<constructor>> ReperfusionNotGiven()	Sets the appropriate superclass values
+ setReperfusionNotGiven(rNotGiven : Byte) : Boolean	Returns true if parameter passes validation rules
+ getReperfusionNotGiven() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ReperfusionNotGiven variable

ReturnExpected	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- returnExpected : Boolean = false	Holds a boolean value, reveals items on UI
+ <<constructor>> ReturnExpected()	Sets the appropriate superclass values
+ setReturnExpected(admASTEMI : Boolean) : Boolean	Sets, then returns returnExpected
+ getReturnExpected() : String	Returns a String value of returnExpected

SmokingStatus	<<extends Value>>
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- smokingStatus : Byte	Holds the short code for patient's smoking status
- smokingStatusLongCode : String	Holds the long code associated with smokingStatus
+ <<constructor>> SmokingStatus()	Sets the appropriate superclass values
+ setSmokingStatus(sStatus : Byte) : Boolean	Returns true if parameter passes validation rules
+ getSmokingStatus() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with smokingStatus variable

SystolicBP	<<**extends Value**>>
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>systolicBP</u> : Double	Holds the patient's systolic blood pressure
+ <<constructor>> SystolicBP()	Set the appropriate super-class values
+ <u>setSystolicBP</u>(sBP : Double) : Boolean	Returns true if parameter passes validation
+ <u>getSystolicBP</u>() : String	Returns systolicBP

Value	<<**superclass**>>
˜com.ucl.appteam7.minapmobile.model	
- id : String	Will contain the dataset field value for each field class
- notes : String	Will contain the field description for each class
- notesTitle : String	Will hold the title for tooltips used in the UI
- isValid : Boolean	To be used in future networking endeavors
- isComplete : Boolean	To be used in future networking endeavors
+ <<constructor>> Value (id : String, notes : String)	Called and populated from field class declarations
+ <<constructor>> Value (id : String, notes : String, notesTitle : String)	Called and populated from field class declarations
+ isValidated() : Boolean	Getter method for isValid variable
+ isCompleted() : Boolean	Getter method for isComplete variable
+ getId() : String	Getter method for id variable
+ getNotes() : String	Getter method for notes variable
+ getNotesTitle() : String	Getter method for notesTitle variable

Weight	<<extends Value>>
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- <u>weight</u> : Double	Holds the patient's weight
+ <<constructor>> Weight()	Set the appropriate super-class values
+ <u>setWeight</u>(w : Double) : Boolean	Returns true if parameter passes validation
+ <u>getWeight</u>() : String	Returns weight

B   Client Communication

Our first contact with the client was on Monday 28th October, with a short meeting outlining the aims of the MINAP project the following day. We have had 3 meetings with the client and other members of MINAP/NICOR so far. All meetings have been recorded (audio) with the consent of those present. A brief summary of the people present and key talking points are outlined below.

B.1   Project Introduction & Requirements (1/11/13)

B.1.1   Meeting Agenda

To get a clear understanding of the project requirements, and to hammer down a realistic project specification that can be delivered within 3 months.

B.1.2   People Present

App Team 7, Lucia Gavalova (Project Manager), Fabian D'Souza (Server Admin), Owen Nicholas, "Tech-guy"

B.1.3   Discussion

1. Feasibility of creating a mobile app from the current web app

    (a) Tech-guy suggests that it is likely that XPages (open-source javascript framework used for business solutions) is the only way of creating an app

    (b) Tech-guy has no idea if the Domino server housing the MINAP data can be accessed via mobile, though suggests XPages as a starting point

2. Fabian called in to hopefully get better ideas on how to implement a mobile solution working with the current technology stack

    (a) Recommends looking at using IBM Lotus Notes (client software to Domino server) as an intermediary for the app

    (b) Suggests that Java could theoretically be used by including a notes.jar file which allows the development of a Lotus Notes plug-in

    (c) Promises to send relevant links that may help with finding a solution

3. Owen and tech-guy suggests that the validation of 130 values of the MINAP dataset will definitely take more than 3 months.

B.1.4 Outcomes

We were thoroughly confused after the meeting with regards to the Domino Server. Fabian did send the e-mail but it only served to confuse even further. We however decided to spend some time researching on the possible ways to communicate with this legacy technology. Lucia provided us with documents containing the full set of data values and validations rules for us to understand. We also provided details to gain access to MINAPs development server to have access to the current web application - this was only processed 2 weeks later.

B.2 UI Feedback (15/11/13)

B.2.1 Meeting Agenda

To obtain client feedback on our preliminary UI, clarify confusions about the MINAP dataset, validation, and use. The target platform needed to be locked down.

B.2.2 People Present

App Team 7, Lucia Gavalova

B.2.3 Discussion

1. Marco David Corrales presented the UI sketches to Lucia

   (a) Lucia is happy with the proposal of a design similar to the current web app

   (b) Mentions that the search functionality can just mirror the simple search in the current web app

   (c) Requests that a tutorial for non-techie users

   (d) David notes down minor comments on UI sketches

2. Team requests a simplified version of the navigation map (and by extension, the 130 value dataset)

   (a) Lucia informs us of her initial motivation for a MINAP mobile app

      i. Mobile app serves as a starting point for medical staff to create a record for patients eligible to be monitored through MINAP when visiting non-cardiac wards

      ii. This helps alleviate the current problem of having positively skewed outcomes as the current patients being recorded are in specialist wards, and more likely to receive treatment when needed

(b) The mobile app could therefore exist with just a subset of the functionality of the existing web app

3. Lucia requests an auto-save feature like Microsoft Word to allow sessions to be resumed later within the same day

4. Amer Kamil Zainal requests if a survey could be designed to get a solid feel of the target audiences primary mobile operating system

(a) Lucia agrees - Kamil promises to design survey

B.2.4 Outcomes

We managed to come to a mutual agreement that the mobile app could serve its main purpose by implementing only a subset of the total MINAP dataset. Lucia promised to send a document detailing the exact values that would need to be included to fulfil this requirement - the document was received some time after the next meeting. A flow diagram of the survey was completed and handed over to Lucia at the next meeting.

B.3 Developer Session (20/11/13)

B.3.1 Meeting Agenda

To understand how the current web app works and how it interacts with the Domino Server; to hand over the device survey flow chart for critique and deployment on SurveyMonkey.

B.3.2 People Present

App Team 7, Lucia Gavalova, Sue Manuel (Developer), Fabian D'Souza

B.3.3 Discussion

1. Sue informs that the web app is made in Javascript, running on top of a Lotus Notes form (in turn, running on the Domino Server)

2. The Javascript methods use the Lotus Notes interface to retrieve data from the database

3. Sue mistakenly tells that validation of values is all server-side - a major setback if correct

4. Fabian called in to verify server-side validation claims - validation is client-side

5. Team request to have a copy of client-side validation code to cross-check with the data validation documents given

6. Discussion with Fabian regarding connecting to Domino Server

   (a) Team confirms research on how to communicate with Domino Server is correct

      i. The Domino Server allows the creation of a web service (consumer and provider)
      ii. This would make the server be compatible with any application that can consume a web service

   (b) Fabian however insists that making our app communicate with a Domino Server is not a key priority, and that spinning off another web server (e.g. Apache + Microsoft SQL Server) with a similar web service capabilities will satisfy the requirements

7. Device survey flow diagram presented and accepted as is by Lucia

B.3.4 Outcomes

The team is now confident in delivering an application with the scaled-down specifications (fewer than 130 validations and Domino Server compatibility not being a priority). A link to the SurveyMonkey device survey was sent to the team, pending approval (by the team). However, due to the reduction of requirements occurring so close to the first report milestone (24/11/13), the team was unable to fully capture this more manageable system in Java.

B.4 Device Survey

Draft proposal: To determine how many target medical personnel own smart-phones (or considering purchase in the near future), and would use it in a work setting. + find out specific OS and screen sizes via requesting visit of a site on their mobile (loaded with Google Analytics).
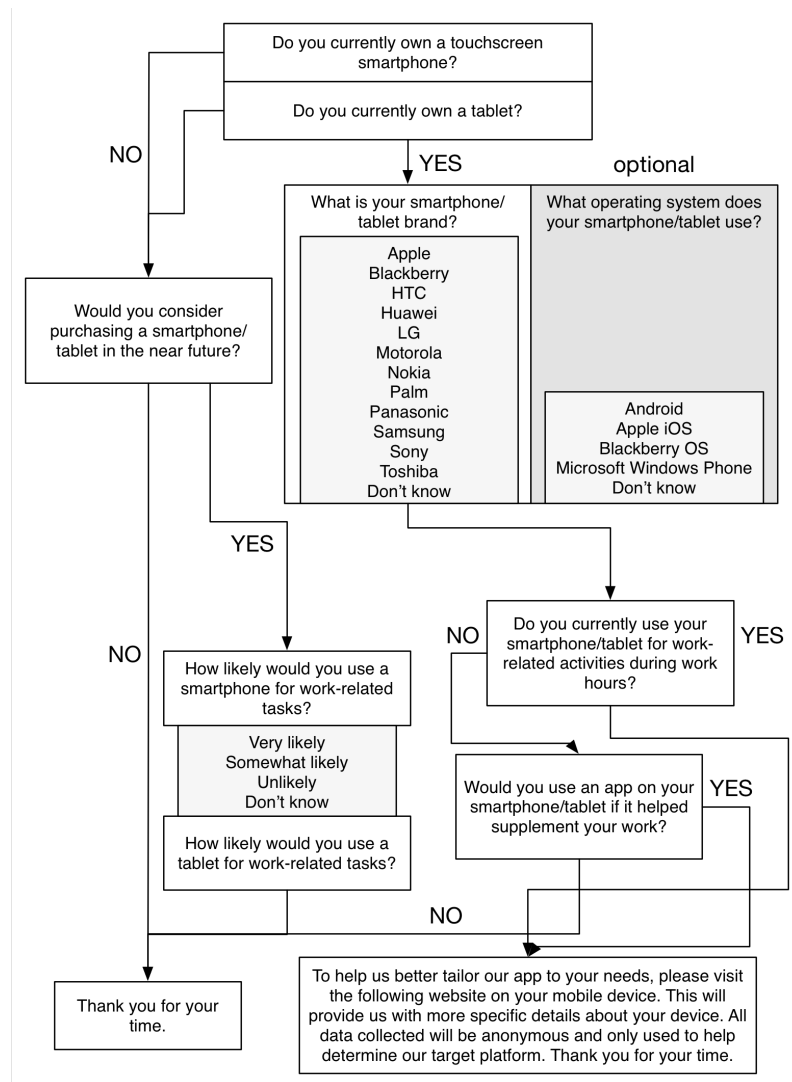

Figure 4: Initial survey flow

It was decided through client feedback that the survey flow logic could be replaced with a simple one-page linear survey due to the straightforward nature of the questions. The contents of the final survey are below.

Final SurveyMonkey Survey

1. Do you currently own a touchscreen smartphone or a tablet?

   (a) Smartphone only

   (b) Tablet only

   (c) Smartphone and tablet

   (d) Neither

2. Would you consider purchasing a smartphone/tablet in the near future?

   (a) Yes

   (b) No

3. What is your smartphone/tablet brand?

   (a) Apple

   (b) Blackberry

   (c) HTC

   (d) Huawei

   (e) LG

   (f) Motorola

   (g) Nokia

   (h) Palm

   (i) Panasonic

   (j) Samsung

   (k) Sony

   (l) Toshiba

   (m) Other (please specify)

   _____

4. What operating system does your smartphone/tablet use (if known)?

   (a) Android

   (b) Apple iOS

   (c) Blackberry OS

   (d) Microsoft Windows Phone

(e) Other (please specify)

5. Do you currently use your smartphone/tablet for work-related activities during work hours?

   (a) Yes

   (b) No

6. Would you use an app on your smartphone/tablet if it helped supplement your work?

   (a) Yes

   (b) No

7. How likely would you be to use a smartphone for work-related tasks?

   (a) Very likely

   (b) Somewhat likely

   (c) Unlikely

   (d) Don't know

8. How likely would you be to use a tablet for work-related tasks?

   (a) Very likely

   (b) Somewhat likely

   (c) Unlikely

   (d) Don't know

9. To help us better tailor our app to your needs, please visit the following website on your mobile device `http://goo.gl/KEevis`. This will provide us with more specific details about your device. All data collected will be anonymous and only used to help determine our target platform. Please feel free to add any comments.

C Internal Meetings and Documentation