## mDefense

## Design

Android's kernel is derived from the Linux kernel. Google contributed code to the Linux kernel as part of their Android effort, but certain features, notably a power management feature called wakelocks, were rejected by mainline kernel developers, so the Android kernel is now a separate version or fork of the Linux kernel.

Google announced in April 2010 that they would hire two employees to work with the Linux kernel community. Greg Kroah-Hartman, the current Linux kernel maintainer for the -stable branch, said in December 2010 that he was concerned that Google was no longer trying to get their code changes included in mainstream Linux. Some Google Android developers hinted that "the Android team was getting fed up with the process", because they were a small team and had more urgent work to do on Android.

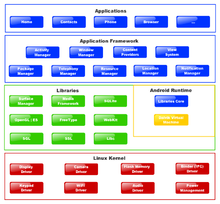
Android does not have a native X Window System nor does it support the full set of standard GNU libraries, and this makes it difficult to port existing GNU/Linux applications or libraries to Android. However, support for the X Window System is possible.

### Features

[](http://en.wikipedia.org/wiki/File:Android_home.png)

[http://bits.wikimedia.org/skins-1.17/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:Android_home.png)

The Android Emulator default home screen (v1.5)

[](http://en.wikipedia.org/wiki/File:Diagram_android.png)

Architecture diagram

Current features and specifications:

Handset layouts

The platform is adaptable to larger, VGA, 2D graphics library, 3D graphics library based on OpenGL ES 2.0 specifications, and traditional smartphone layouts.

Storage

SQLite, a lightweight relational database, is used for data storage purposes.

Connectivity

Android supports connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.

Messaging

SMS and MMS are available forms of messaging, including threaded text messaging and now Android Cloud To Device Messaging Framework(C2DM) is also a part of Android Push Messaging service.

Multiple language support

Android supports multiple human languages. The number of languages more than doubled for the platform 2.3 Gingerbread. Android lacks font rendering of several languages even after official announcements[of added support (e.g. Hindi).

Web browser

The web browser available in Android is based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine. The browser scores a 93/100 on the Acid3 Test.

Java support

While most Android applications are written in Java, there is no Java Virtual Machine in the platform and Java byte code is not executed. Java classes are compiled into Dalvik executables and run on Dalvik, a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU. J2ME support can be provided via third-party applications.

Media support

Android supports the following audio/video/still media formats: WebM, H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, FLAC, WAV, JPEG, PNG, GIF, BMP.

Streaming media support

RTP/RTSP streaming (3GPP PSS, ISMA), HTML progressive download (HTML5 <video> tag). Adobe Flash Streaming (RTMP) and HTTP Dynamic Streaming are supported by the Flash plugin. Apple HTTP Live Streaming is supported by RealPlayer for Mobile, and by the operating system in Android 3.0 (Honeycomb).

Additional hardware support

Android can use video/still cameras, touchscreens, GPS, accelerometers, gyroscopes, magnetometers, dedicated gaming controls, proximity and pressure sensors, thermometers, accelerated 2D bit blits (with hardware orientation, scaling, pixel format conversion) and accelerated 3D graphics.

Multi-touch

Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero. The feature was originally disabled at the kernel level (possibly to avoid infringing Apple's patents on touch-screen technology at the time). Google has since released an update for the Nexus One and the Motorola Droid which enables multi-touch natively.

Bluetooth

Supports A2DP, AVRCP, sending files (OPP), accessing the phone book (PBAP), voice dialing and sending contacts between phones. Keyboard, mouse and joystick (HID) support is available in Android 3.1+, and in earlier versions through manufacturer customizations and third-party applications.

Video calling

Android does not support native video calling, but some handsets have a customized version of the operating system that supports it, either via the UMTS network (like the Samsung Galaxy S) or over IP. Video calling through Google Talk is available in Android 2.3.4 and later. Gingerbread allows Nexus S to place Internet calls with a SIP account. This allows for enhanced VoIP dialing to other SIP accounts and even phone numbers. Skype 2.1 offers video calling in Android 2.3, including front camera support.

Multitasking

Multitasking of applications is available.

Voice based features

Google search through voice has been available since initial release. Voice actions for calling, texting, navigation, etc. are supported on Android 2.2 onwards.

Tethering

Android supports tethering, which allows a phone to be used as a wireless/wired Wi-Fi hotspot. Before Android 2.2 this was supported by third-party applications or manufacturer customizations.

Screen capture

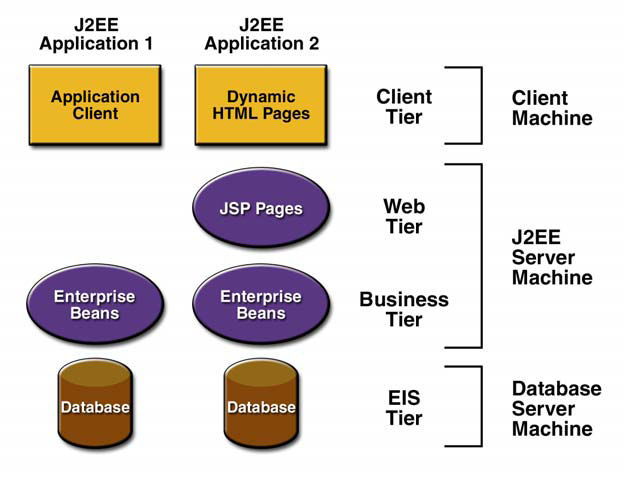
Android does not support screenshot capture as of 2011. This is supported by manufacturer and third-party customizations. Screen Capture is available through a PC connection using the DDMS developer's tool.

**4.5 J2EE (Java 2 Platform, Enterprise Edition)**

* Introduction
* Distributed Multi-tiered Applications
* J2EE Components
* Conclusion

The J2EE platform uses a distributed multitiered application model for enterprise applications. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs. Multitiered J2EE applications divided into the tiers described in the following list:

* Client-tier components run on the client machine.
* Web-tier components run on the J2EE server.
* Business-tier components run on the J2EE server.
* Enterprise information system (EIS)-tier software runs on the EIS server.



**Multi-tiered J2EE Application**

**J2EE Components:**

J2EE applications are made up of components. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components. The J2EE specification defines the following J2EE components:

* Application clients and applets are components that run on the client.
* Java Servlet and Java Server Pages™ (JSP™) technology components are Web components that run on the server.
* Enterprise JavaBeans™ (EJB™) components (enterprise beans) are business components that run on the server.

J2EE web components are either servlets or pages created using JSP technology (JSP pages). Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that execute as servlets but allow a more natural approach to creating static content. Static HTML pages and applets are bundled with web components during application assembly but are not considered web components by the J2EE specification. Server-side utility classes can also be bundled with web components and, like HTML pages, are not considered web components.

**4.6 Web service**

Web services constitute a distributed computer architecture made up of many different computers trying to communicate over the network to form one system. They consist of a set of standards that allow developers to implement distributed applications - using radically different tools provided by many different vendors - to create applications that use a combination of software modules called from systems in disparate departments or from other companies.

A Web service contains some number of classes, interfaces, enumerations and structures that provide black box functionality to remote clients. Web services typically define business objects that execute a unit of work (e.g., perform a calculation, read a data source, etc.) for the consumer and wait for the next request. Web service consumer does not necessarily need to be a browser-based client. Console-based and Windows Forms-based clients can consume a Web service. In each case, the client indirectly interacts with the Web service through an intervening proxy. The proxy looks and feels like the real remote type and exposes the same set of methods. Under the hood, the proxy code really forwards the request to the Web service using standard HTTP or optionally SOAP messages.

**Web Service Standards:**

Web services are registered and announced using the following services and protocols. Many of these and other standards are being worked out by the UDDI project, a group of industry leaders that is spearheading the early creation and design efforts. Universal Description, Discovery, and Integration (UDDI) is a protocol for describing available Web services components. This standard allows businesses to register with an Internet directory that will help them advertise their services, so companies can find one another and conduct transactions over the Web. This registration and lookup task is done using XML and HTTP(S)-based mechanisms.

Simple Object Access Protocol (SOAP) is a protocol for initiating conversations with a UDDI Service. SOAP makes object access simple by allowing applications to invoke object methods or functions, residing on remote servers. A SOAP application creates a request block in XML, supplying the data needed by the remote method as well as the location of the remote object itself.

Web Service Description Language (WSDL), the proposed standard for how a Web service is described, is an XML-based service IDL (Interface Definitition Language) that defines the service interface and its implementation characteristics. WSDL is referenced by UDDI entries and describes the SOAP messages that define a particular Web service.

EbXML (e-business XML) defines core components, business processes, registry and repository, messaging services, trading partner agreements, and security.

**CHAPTER-5**

**DESIGN**

Design Patterns brought a paradigm shift in the way object oriented systems are designed. Instead of relying on the knowledge of problem domain alone, design patterns allow past experience to be utilized while solving new problems. Traditional object oriented design (OOD) approaches such as Booch, OMT, etc. advocated identification and specification of individual objects and classes. Design Patterns on the other hand promote identification and specification of collaborations of objects and classes. However, much of the focus of recent research has been towards identification and cataloging of new design patterns. The effort has been to assimilate knowledge gained from designing systems of the past, in various problem domains. The problem analysis phase has gained little benefit from this paradigm. Most projects still use traditional object oriented analysis (OOA) approaches to identify classes from the problem description. Responsibilities to those classes are assigned based upon the obvious description of entities given in the problem definition.

Pattern Oriented Technique (POT) is a methodology for identifying interactions among classes and mapping them to one or more design patterns. However, this methodology also uses traditional OOA for assigning class responsibilities. As a result, its interaction oriented design phase (driven by design patterns) receives its input in terms of class definitions that might not lead to best possible design.

The missing piece here is the lack of an analysis method that can help in identifying class definitions and the collaborations between them which would be amenable to application of interaction oriented design. There are two key issues here. First is to come up with good class definitions and the second is to identify good class collaborations.

It has been observed in that even arriving at good class definitions from the given problem definition is non-trivial. The key to various successful designs is the presence of abstract classes (such as an event handler) which are not modeled as entities in the physical world and hence do not appear in the problem description. In anticipating change has been proposed as the method for identifying such abstract classes in a problem domain. Another difficult task is related to assignment of responsibilities to entities identified from the problem description. Different responsibility assignments could lead to completely different designs. Current approaches such as Coad and Yourdon, POT etc. follow the simple approach of using entity descriptions in the problem statement to define classes and fix responsibilities. We propose to follow a flexible approach towards assigning responsibilities to classes so that the best responsibility assignment can be chosen.

The second issue is to identify class collaborations. Techniques such as POT analyze interactions among different sets of classes as specified in the problem description. Such interacting classes are then grouped together to identify design patterns that may be applicable. However, as mentioned earlier, only the interactions among obvious classes are determined currently. Other interactions involving abstract classes not present in the problem or interactions that become feasible due to different responsibility assignments are not considered. We present some techniques that enable the designer to capture such interactions as well.

**INTERACTION BASED ANALYSIS AND DESIGN**

**Top-down approach:**

This approach is applicable to situations where the designer knows the solution to the given problem. It is true for problem domains that have well established high-level solutions and different implementations vary in low level details (for e.g. Enterprise Resource Planning (ERP) systems). Her main concern is to realize that solution in a way such that the implemented system has nice properties such as maintainability and reusability etc.

To achieve this goal, the system designer selects appropriate design patterns that form the building blocks of her solution. Having obtained this design template (design type), she maps the classes and objects participating in those patterns to the entities of the problem domain. This mapping implicitly defines the responsibilities of various classes/objects that represent those entities. To help clarify the concept, consider a scenario where an architect is assigned the task of building a flyover. Flyover construction is an established science and the architect knows the solution to the problem. She starts by identifying component patterns such as road strip, support pillars, side railings and so on. Having done that, she maps the participating objects to actual entities in the problem domain. This would involve defining the length and width of the road strip based upon the space constraints specified in the problem. The height and weight of the pillars get decided based upon the load requirements specified. The entry and exit points get decided based upon the geography of the location and so on. This results in a concrete design instance. Some new classes or objects, not existing in the domain model, may also have to be introduced for a successful instantiation of the design template. For instance, the problem domain may not model an abstract entity such as an event handler which may be a participant in some portion of the design template. Such generic classes/objects may be drawn from a common repository of utility classes. Interaction driven analysis phase here is simple since the interactions (in the form of design patterns) are already well established and directly obtained from the knowledge base.

**Bottom -up approach:**

This approach is applicable in scenarios where interactions in the problem domain are not well understood and need to be discovered and explored. This situation is a fundamental problem faced by the designers of object oriented systems. It relates to the fact that objects oriented analysis (OOA) does not help much in creating a solution to the problem at hand. The analysis phase is mainly concerned with enhancing the understanding of the problem domain. This knowledge is then later used by a problem solving approach to come up with a solution possessing good design properties. As a result, at the end of the analysis phase the designer has a set of well defined components that need to be assembled together for realizing a solution. For instance, to build a route finder application the OOA phase helps in modeling the domain objects such as roads, vehicles, cities, addresses etc. but does not actually provide a solution for finding routes between two given addresses. This is similar to having various pieces of a jigsaw puzzle but the puzzle still needs to be solved. The problem in software systems is further complicated by the fact that there is generally no unique solution to a problem. There are always trade-offs at various stages and the resulting designs are a reflection of the choices made at those stages. In the jigsaw puzzle example this is similar to the situation where different sets of the same puzzle are available each differing from another in terms of the design of its component pieces. Some component designs may help in solving the puzzle faster and more efficiently than others.

The bottom-up approach helps in such situations where the entities in the problem domain have been identified by traditional OOA techniques but multiple choices exist in terms of assigning responsibilities to those entities. Unlike top-down approach, the mapping of responsibilities to entities is not dictated by the design solution specified by the designer. Instead, the task of the designer here is to try various responsibility assignments and create an interaction specification involving those objects. The objective of this interaction driven analysis is to obtain an interaction specification that helps in arriving at a solution with best design characteristics possible. Having identified the entities in the domain, the starting point for the designer is to identify various alternatives available for assigning responsibilities to individual objects. Her domain knowledge helps her in this task. Given these alternatives for potential object definitions and standard utility objects (such as schedulers, event handlers etc.), the next step is to find compositions of these building blocks (i.e. interactions of these objects) that provide alternative solutions to the problem. This task is a non-trivial one especially when done manually. There are just too many combinations to be considered, for any human designer to obtain alternative solutions in a reasonable amount of time. We need to apply semi-automated software composition techniques based on some formal specification. Several such approaches have been recently investigated in the context of e-services. These include workflow based approaches and AI Planning based techniques. Other formal techniques for specifying composition include Petri-net based models, automata-based models; temporal logics etc. from verification community and X Query, XML constraint tools based techniques from data management community.

The resulting candidate compositions (i.e. interaction specifications) then need to be compared with existing design patterns either manually or automatically. It is not beyond imagination to visualize that with advancement in automated composition techniques, new design patterns may get identified during this process. For instance, techniques such as Reinforcement Learning have resulted in new novel solutions in various domains such as playing Backgammon. In such a case, the resulting designs may need to be evaluated manually. The best design among the alternatives is then chosen for implementing the system.

**Language for Specifying Design Patterns:**

The approaches for OO Design proposed in this paper favor automatic techniques over manual ones for reasons described earlier. This means that we need a mechanism to be able to express design patterns in a format amenable to be read and interpreted by programs. Some attempts have been made at defining such pattern description languages. One of these or some variation of these could be used to express design patterns in a formal language.

**Comparison of Software Designs:**

Once we have alternative designs available, they need to be compared to arrive at the best one.

Each design may consist of multiple design patterns. The criteria here would not be to simply count the number of design patterns used but to evaluate the interaction between patterns and also between other design elements used. This would involve an understanding of good and bad design interactions and an ability to identify them in a given design. The final challenge would be to do it automatically.

**5.1 Structural Diagram**

**Class Diagram:**

Class diagrams identify the class structure of a system, including the properties and methods of each class. Also depicted are the various relationships that can exist between classes, such as an inheritance relationship.

****

FIGURE 5.1.1 SHOWING USER, ADMINISTRATOR, APRIORI CLASSES SHOWING THEIR CORRESPONDING ATTRIBUTES AND OPERATIONS

**Object diagram:**

Object diagrams model instances of classes. This type of diagram is used to describe the system at a particular point in time. Using this technique, you can validating the class diagram and it's multiplicity rules with real-world data, and record test scenarios. From a notation standpoint, Object diagrams borrow elements from class diagrams.

**5.2 Component Diagram:**

Component diagrams fall under the category of an implementation diagram, a kind of diagram that models the implementation and deployment of the system. A Component Diagram, in particular, is used to describe the dependencies between various software components such as the dependency between executable files and source files. This information is similar to that within make files, which describe source code dependencies and can be used to properly compile an application.

**5.3 Deployment Diagram**

Deployment diagrams are another model in the implementation diagram category. The Deployment diagram models the hardware used in implementing a system and the association between those hardware components. Components can also be shown on a Deployment diagram to show the location of their deployment. Deployment diagrams can also be used early on in the design phase to document the physical architecture of a system.

**5.4 Behavioral Diagrams**

**Use Case Diagram:**

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of Use Case diagrams include:

* Providing a high-level view of what the system does
* Identifying the users ("actors") of the system
* Determining areas needing human-computer interfaces

Use Cases extend beyond pictorial diagrams. In fact, text-based use case descriptions are often used to supplement diagrams, and explore use case functionality in more detail.

****

**FIGURE 5.4.1 USECASES SHOWING ANDROID,FLEX APPLICATIONS AND THEIR FUNCTIONALITY**

**5.5 Sequence Diagram**

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.



FIGURE 5.5.1 SEQUENCE DIAGRAM SHOWING OBJECTS AND THEIR CORRESPONDING MESSAGES

**5.6 Collaboration Diagram**

Collaboration diagrams model the interactions between objects. This type of diagram is a cross between an object diagram and a sequence diagram. It uses free-form arrangement of objects which makes it easier to see all iterations involving a particular object.

FIGURE 5.6.1 COLLOBRATION DIAGRAM SHOWING INTERACTION BETWEEN OBJECTS

**5.7 State chart Diagram** State diagrams, are used to document the various modes ("State") that a class can go through, and the events that cause a state transition.

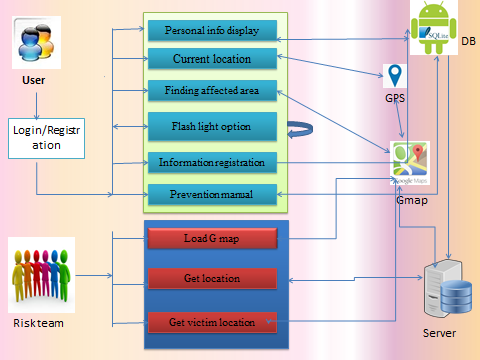
**5.8 Activity Diagram**

Activity diagrams are used to document workflows in a system, from the business level down to the operational level. The general purpose of Activity diagrams is to focus on flows driven by internal processing vs. external events.

**CHAPTER-6**

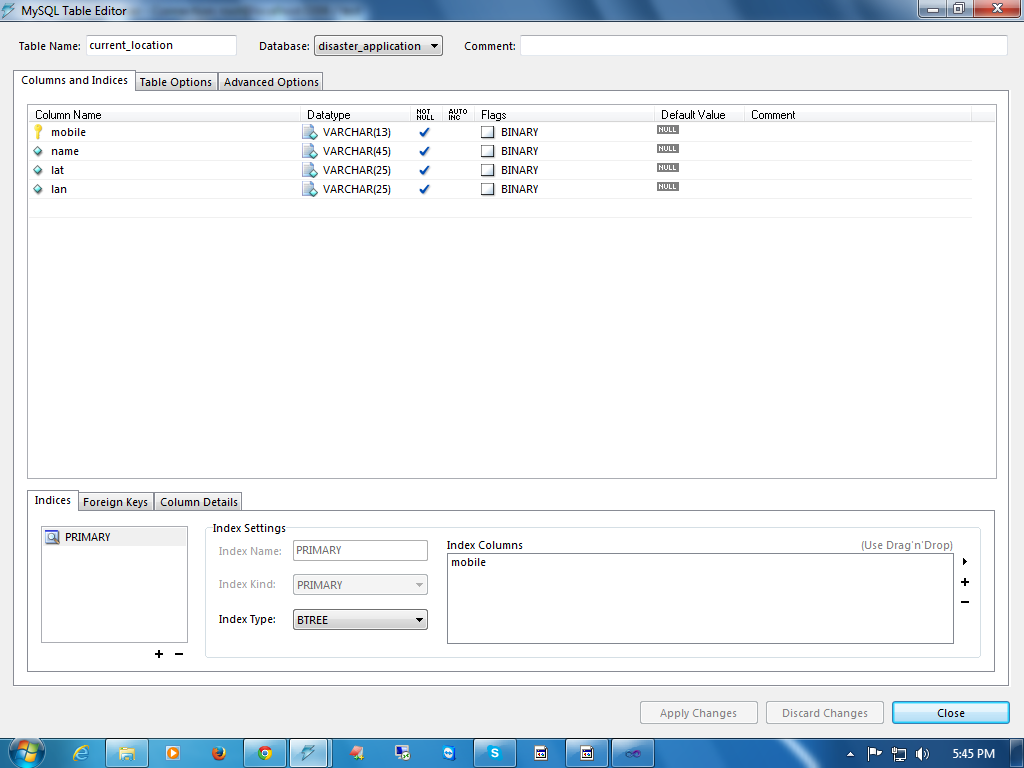
**ARCHITECTURE**

Project architecture represents no of components we are using as part of our project and the flow of request processing.

****

**CHAPTER-7**

**DATA BASE**



CHAPTER-8

PROGRAMMING CODE

package in.genesissol.dspapp4useincaseofdisaster;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import in.genesissol.database.DataBaseOperations;

import in.genesissol.database.Record;

import in.genesissol.dspapp4useincaseofdisaster.util.AlertWindow;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.Intent;

import android.database.Cursor;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.util.Log;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.AdapterView;

import android.widget.AdapterView.OnItemSelectedListener;

import android.widget.ArrayAdapter;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageView;

import android.widget.Spinner;

import android.widget.TextView;

import android.widget.Toast;

public class PersonalInformationActivity

extends Activity

implements OnClickListener,

OnItemSelectedListener{

private EditText name;

private EditText father;

private EditText address;

private EditText mobile;

private EditText family\_doctor;

private EditText blood\_group;

private EditText contact;

private TextView lbl\_Update;

private Spinner diseases;

private Button upDate;

private ImageView photo;

private Record record;

private DataBaseOperations dbo;

private ArrayAdapter<String> adapter;

@SuppressWarnings("unused")

private Bitmap bitmap;

@SuppressWarnings("unused")

private int selectionCurrent;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_personal\_information);

loadViews();

dbo=new DataBaseOperations(this);

record= dbo.getProfile();

addListeners();

if(record!=null){

loadValues(record);

}

else{

Toast.makeText(getApplicationContext(), "Register Profile first Time",Toast.LENGTH\_LONG).show();

enableValues();

lbl\_Update.setVisibility(View.INVISIBLE);

upDate.setVisibility(View.VISIBLE);

updateProfile();

}

}

private void updateProfile() {

// TODO Auto-generated method stub

record=getValues();

if(dbo.createProfile(record)){

Log.i("status","successfully inserted");

loadValues(record);

}

else{

Log.i("status","insertion fail");

}

}

private Record getValues() {

// TODO Auto-generated method stub

Record record=new Record();

record.setName(name.getEditableText().toString());

record.setFather(father.getEditableText().toString());

record.setAddress(address.getEditableText().toString());

record.setMobile(mobile.getEditableText().toString());

record.setFamily\_doctor(family\_doctor.getEditableText().toString());

record.setBlood\_group(blood\_group.getEditableText().toString());

record.setContact(contact.getEditableText().toString());

record.setPhoto(new byte[0]);

return record;

}

private void loadValues(Record rcrd) {

// TODO Auto-generated method stub

// if(rcrd.getPhoto())

// bitmap = BitmapFactory.decodeByteArray(rcrd.getPhoto() , 0,rcrd.getPhoto() ;

// photo.setImageBitmap(bitmap);

name.setText(rcrd.getName());

father.setText(rcrd.getFather());

address.setText(rcrd.getAddress());

mobile.setText(rcrd.getMobile());

family\_doctor.setText(rcrd.getFamily\_doctor());

blood\_group.setText(rcrd.getBlood\_group());

contact.setText(rcrd.getContact());

adapter=new ArrayAdapter<String>(this,android.R.layout.simple\_spinner\_item, dbo.getDiseases());

diseases.setAdapter(adapter);

disableValues();

}

public void addListeners() {

// TODO Auto-generated method stub

lbl\_Update.setOnClickListener(this);

upDate.setOnClickListener(this);

photo.setOnClickListener(this);

diseases.setOnItemSelectedListener(this);

}

public void loadViews(){

name=(EditText) findViewById(R.id.name);

father=(EditText) findViewById(R.id.father);

address=(EditText) findViewById(R.id.address);

mobile=(EditText) findViewById(R.id.mobile);

family\_doctor=(EditText) findViewById(R.id.doctor);

blood\_group=(EditText) findViewById(R.id.blood);

contact=(EditText) findViewById(R.id.contact);

lbl\_Update=(TextView) findViewById(R.id.lbl\_edit);

diseases=(Spinner) findViewById(R.id.diseases);

upDate=(Button) findViewById(R.id.update);

photo=(ImageView) findViewById(R.id.photo);

photo.setVisibility(View.INVISIBLE);

upDate.setVisibility(View.INVISIBLE);

lbl\_Update.setVisibility(View.VISIBLE);

}

@Override

public void onClick(View v) {

// TODO Auto-generated method stub

switch(v.getId()){

case R.id.lbl\_edit:

lbl\_Update.setVisibility(View.INVISIBLE);

upDate.setVisibility(View.VISIBLE);

enableValues();

break;

case R.id.update:

lbl\_Update.setVisibility(View.VISIBLE);

upDate.setVisibility(View.INVISIBLE);

updateProfile();

break;

case R.id.photo:

if(upDate.isShown())

{

Toast.makeText(getApplicationContext(), "select image from sd card",Toast.LENGTH\_LONG).show();

readPhoto();

}

break;

}

}

private void readPhoto() {

// TODO Auto-generated method stub

Intent intent = new Intent(Intent.ACTION\_GET\_CONTENT, null);

intent.setType("image/\*");

intent.putExtra("return-data", true);

startActivityForResult(intent, 1);

}

@Override

public void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

switch (requestCode) {

case 1:

if(requestCode == 1 && data != null && data.getData() != null){

Uri \_uri = data.getData();

if (\_uri != null) {

//User had pick an image.

Cursor cursor = getContentResolver().query(\_uri, new String[] { android.provider.MediaStore.Images.ImageColumns.DATA }, null, null, null);

cursor.moveToFirst();

//Link to the image

final String imageFilePath = cursor.getString(0);

Log.v("imageFilePath", imageFilePath);

File photos= new File(imageFilePath);

Bitmap b = decodeFile(photos);

b = Bitmap.createScaledBitmap(b,150, 150, true);

/\* ImageView imageView = (ImageView) findViewById(R.id.select\_image);

imageView.setImageBitmap(b);\*/

photo.setImageBitmap(b);

cursor.close();

}

}

super.onActivityResult(requestCode, resultCode, data);

}

}

private void enableValues() {

// TODO Auto-generated method stub

name.setEnabled(true);

father.setEnabled(true);

address.setEnabled(true);

mobile.setEnabled(true);

family\_doctor.setEnabled(true);

blood\_group.setEnabled(true);

contact.setEnabled(true);

}

private void disableValues() {

// TODO Auto-generated method stub

name.setEnabled(false);

father.setEnabled(false);

address.setEnabled(false);

mobile.setEnabled(false);

family\_doctor.setEnabled(false);

blood\_group.setEnabled(false);

contact.setEnabled(false);

}

@Override

public void onItemSelected(AdapterView<?> arg0, View arg1, int position,long arg3) {

// TODO Auto-generated method stub

if(upDate.isShown())

AlertWindow.recordDisease(PersonalInformationActivity.this);

selectionCurrent= position;

}

@Override

public void onNothingSelected(AdapterView<?> arg0) {

// TODO Auto-generated method stub

}

private Bitmap decodeFile(File f){

try {

//decode image size

BitmapFactory.Options o = new BitmapFactory.Options();

o.inJustDecodeBounds = true;

BitmapFactory.decodeStream(new FileInputStream(f),null,o);

//Find the correct scale value. It should be the power of 2.

final int REQUIRED\_SIZE=20;

int width\_tmp=o.outWidth, height\_tmp=o.outHeight;

int scale=1;

while(true){

if(width\_tmp/2<REQUIRED\_SIZE || height\_tmp/2<REQUIRED\_SIZE)

break;

width\_tmp/=2;

height\_tmp/=2;

scale++;

}

//decode with inSampleSize

BitmapFactory.Options o2 = new BitmapFactory.Options();

o2.inSampleSize=scale;

return BitmapFactory.decodeStream(new FileInputStream(f), null, o2);

} catch (FileNotFoundException e) {}

return null;

}

**OUTPUT SCREENS**



