Shanghai Jiao Tong University

Android Programming

Zhenhao Cao

April 8, 2018

Contents

1 Introduction **2**

2 Environment **2**
2.1 Installation and Configuration of JDK 2
2.2 Installation and Configuration of Android SDK 3
2.3 About Eclipse and ADT 4
2.4 Android Studio 4

3 Android Programming **5**
3.1 "Hello, world!" 5
3.1.1 AndroidManifest.xml 5
3.1.2 mainActivity_layout.xml 6
3.1.3 mainActivity.java 7
3.2 App Contact 7
3.2.1 Onclick Listener 7
3.2.2 Activity Switch 8
3.2.3 Data Transmission 8
3.2.4 SMS Sending 9

4 Experiment Result **9**
4.1 Use Case 1: Add new contacts 10
4.2 Use Case 2: Write a message 11

5 Discussion **12**
5.1 Customized Adapter 12
5.2 Transmission of A List 13

1 Introduction

This report aims to present cardinal information about Android Programming and introduce our work at this lab.

Apps are everywhere now. As a type of high-level functional software, these ‘little icons’ have played important roles in communication, entertainment, consumption and even academia in our modern life. Based on Java, Android programming has been developed into a relatively mature technique to implement software functions as well as provide users with friendly interfaces. switches, usage of common widgets, and command of basic java classes and functions. We implement a delicate self-made contact app.

The remainder of this report is organized as follows. Section 2 gives an illustration of the environment needed by Android programming. In Section 3, we comprehensively show our detailed work in this lab. In Section 4, pictorial experiment results are presented. In Section 5, we further discuss hard problems we encountered in this lab.

2 Environment

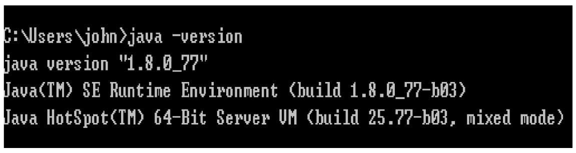
This section gives a brief description of the environment of this lab.

2.1 Installation and Configuration of JDK

JDK (Java Development Kit) refers to the software developing toolbox of Java. It has been generally used for Java applications on mobile and embedded devices. JDK is the core component of Java developing. The JDK includes a private JVM and a few other resources to finish the development of a Java Application.

To install the JDK, we should choose the correct version (matching the computer system) and download it first. After executing the installing program, we should configure the environment variables to include its current directory.

By inputting ’java -version’ at cmd, we can test whether the JDK is installed and configured correctly. Version information the current JDK will be shown if correctly installed.



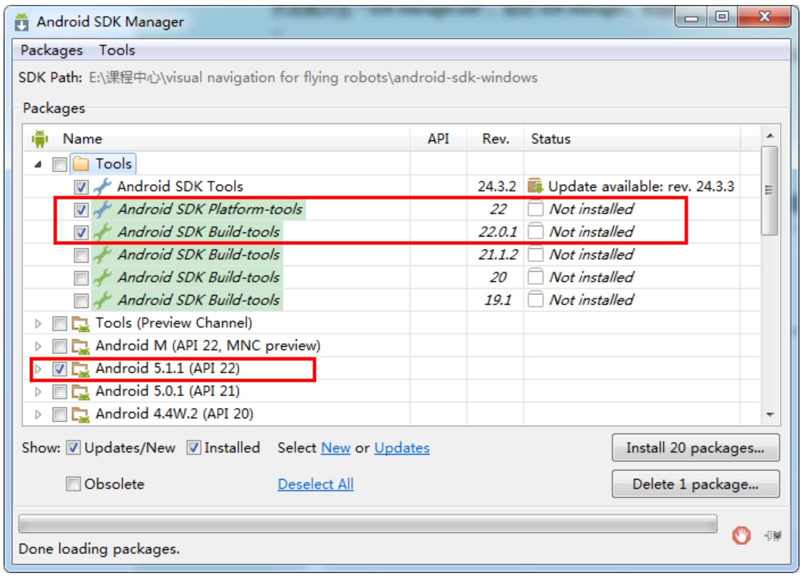
```
C:\Users\john>java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Figure 1: Test of JDK

2.2 Installation and Configuration of Android SDK

Officially, apps can be written using Java, C++ or Kotlin using the Android Software Development Kit (SDK). The Android Software Development Kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU and etc.

To install Android SDK, we should download the installation package of the correct version from official website. After unzipping the package, we should execute the SDK Manager and check all the tools needed. Also, API of the latest version should be checked and downloaded. Thereafter, we should configure the environment variables by including the current path of SDK.



SDK Path: E:\课程中心\visual navigation for flying robots\android-sdk-windows

Packages

Name	API	Rev.	Status
☐ Tools			
☑ Android SDK Tools		24.3.2	Update available: rev. 24.3.3
☑ Android SDK Platform-tools		22	Not installed
☑ Android SDK Build-tools		22.0.1	Not installed
☐ Android SDK Build-tools		21.1.2	Not installed
☐ Android SDK Build-tools		20	Not installed
☐ Android SDK Build-tools		19.1	Not installed
☐ Tools (Preview Channel)			
☐ Android M (API 22, MNC preview)			
☑ Android 5.1.1 (API 22)			
☐ Android 5.0.1 (API 21)			
☐ Android 4.4W.2 (API 20)			

Show: ☑ Updates/New ☑ Installed Select New or Updates

☐ Obsolete Deselect All

Done loading packages.

Figure 2: Installation of Android SDK

2.3 About Eclipse and ADT

The combination of Eclipse and ADT is known as the last-generation developing platform for Android programming. Still, some developers stick to this combination nowadays. After installing and configuring JDK and SDK, an installing package of Eclipse should be downloaded from the official website of Eclipse. Execute the installation program then. Thereafter, install the ADT plug-in online for Eclipse. Restart Eclipse and create an emulator, then we can start programming in a new project.

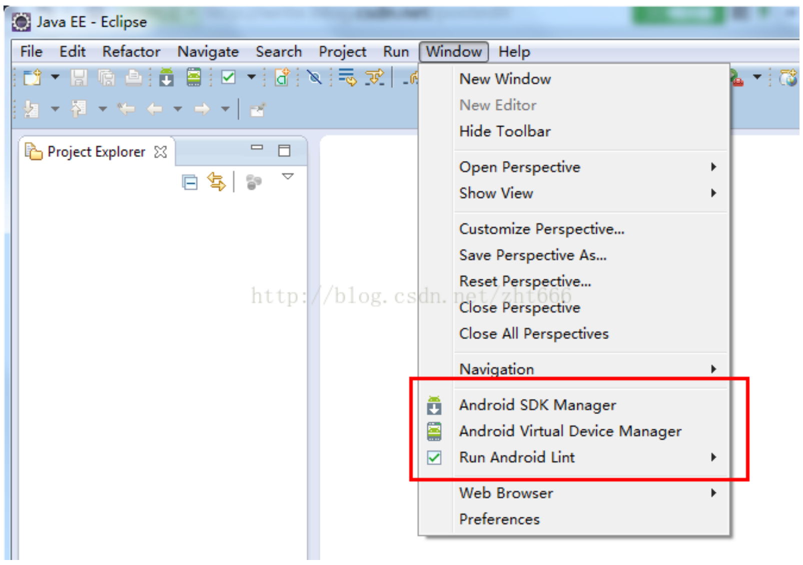
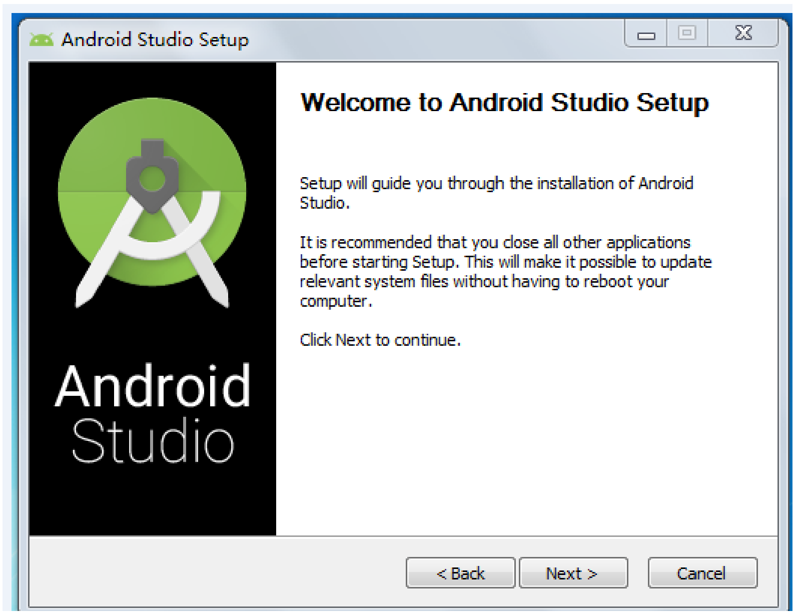


Figure 3: Use Eclipse with ADT

2.4 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

Android Studio can be easily accessed, installed and configured following official instruction. We can start programming after creating a new project. Frequently-used templates are pre-defined by Android Studio and we can base them to program.



Welcome to Android Studio Setup

Setup will guide you through the installation of Android Studio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 4: Installation of Android Studio

3 Android Programming

This section describes the process we accomplish the detailed tasks in Lab 1, including the presence of "Hello, world" and the implementation of App Contact. In the latter task, we additionally add SMS sending function to the Contact app.

3.1 "Hello, world!"

This is the elementary part of Lab 1, which requires us to show "Hello, world!" on a emulator with Android programming. This task enables us to get familiar with the basic usage of Android activity and TextView widget (or other possible widget). In this task, only a main activity and a TextView widget are needed to present the desired effect. The final effect is shown in Fig.5.

3.1.1 AndroidManifest.xml

Before modifying the java class file of the main activity, we should register it in the manifest file. Note that if the target activity is a main activity, we should configure the content in label *intent-filter*; otherwise this is unnecessary. The label *Name*

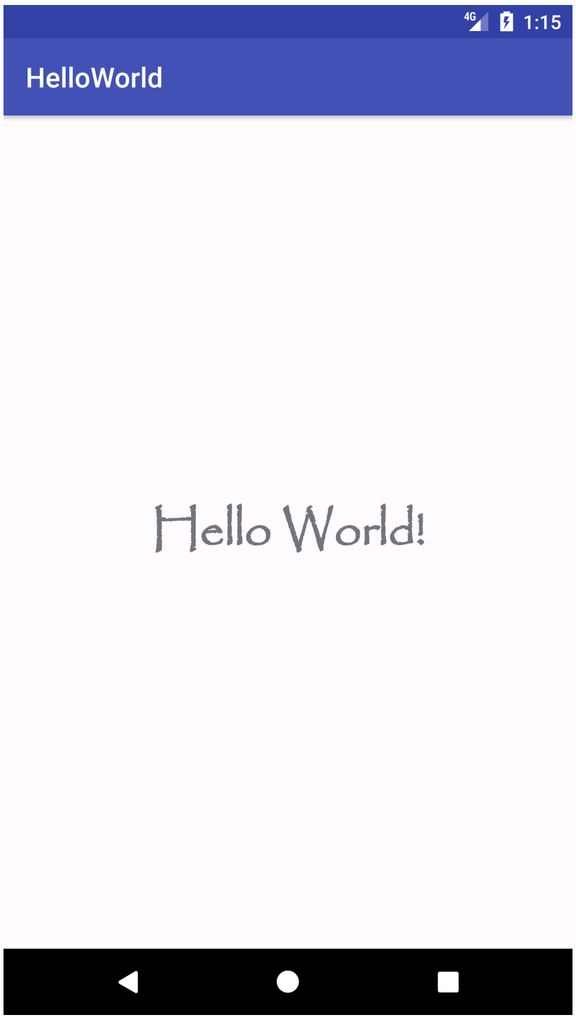


Figure 5: Hello, world!

specifies the target activity, and *Label* specifies both the content in title bar and the name of our application in Launcher.

3.1.2 mainActivity_layout.xml

Layout files (.xml) are used to design and build basic page frames for activities. We commonly set all necessary widgets directly in layout files. Note that widgets are fixed in a layout in forms of embedding, with a parent container (layout widdet) set beforehand. In this elementary task, we might try *RelativeLayout* as well. Widgets in *RelativeLayout* locate themselves by their parents layout or other widgets in the same page. We use TextView to show ”Hello, world!” in this experiment. Two of the available attributes for location in *RelativeLayout* include the follows.

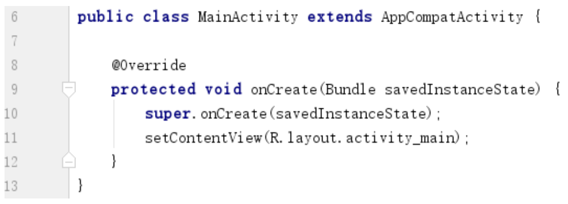
$$android : layout\_alighParentTop$$

$$android : layout\_alighParentLeft$$

Frequently-used attributes of TextView include *text*, *textSize*, *textColor*, *textStyle* and *typeFace* etc., controlling all kinds of styles of TextView widget.

3.1.3 mainActivity.java

Java class files are used to launch target activities and implement more complex functions apart from basic layouts. Any required activity needs to be launched explicitly in the corresponding java file. Other java files are used to define specific classes, including logic classes (arrayList, item etc.) and widget classes (compound widgets) for multiplexing. In the first task, we only use java file as activity implementation file. Necessary contents are shown as follow.



```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Figure 6: Activity launching

3.2 App Contact

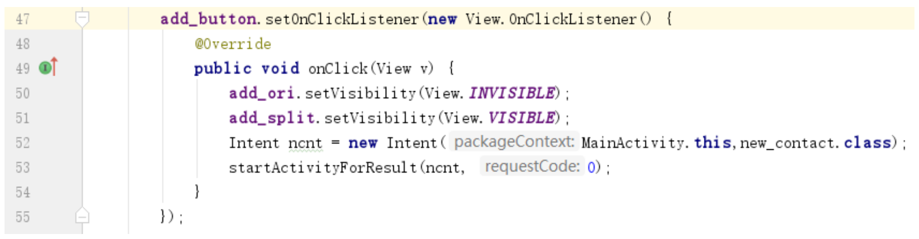
This is the advanced part of Lab 1, which requires us to implement a Contact App and make modifications. This task enables us to master more complex functions provided by Android programming, including the monitoring of onclicks and corresponding reaction. Activity switches, data transmission and SMS sending are also involved in this experiment.

3.2.1 Onclick Listener

While using an App, we must have noticed this common scenario that some reaction (e.g. page switch) is triggered by clicking a button. This is actually implemented by an onclick-monitor machnism in Android programming. The monitor is call ‘listener’ in this context. Note that we specify the widget by tracking its *id*, and then bind an onclick-listener on it. In listener funcion we override the orginal *onClick* function with a customized one, in which we set reacting functions.

3.2.2 Activity Switch

One of the most common reaction while clicking a button is page switching. This is actually implemented by activity calling and switching in Andoird programming. Intuitively, we independently implement the internal logic and external layouts of activities. Then how can we connect different activities (pages) and make them call each other? The built-in class *intent* can help us accomplish this. We use an *intent* to clarify the calling relationship beteen two activities (from whom to whom) then use *startActivity* function (or *startActivityForResult* if result data are expected) to switch to a new activity (page). A typical implementation of activity switch triggered by an onclick listener in our project is shown as Fig.7.

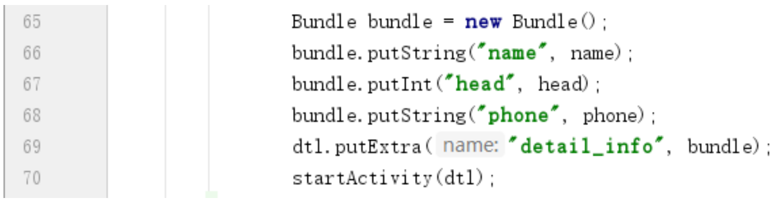


```
add_button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        add_ori.setVisibility(View.INVISIBLE);
        add_split.setVisibility(View.VISIBLE);
        Intent ncnt = new Intent( packageContext: MainActivity.this,new_contact.class);
        startActivityForResult(ncnt,  requestCode: 0);
    }
});
```

Figure 7: Onclick listener

3.2.3 Data Transmission

Data transmission between activities are necessary. Take our Contact App as an example, after clicking a contact item on main page, a new page showing detailed information of the contact will be triggered. In this process, the ID of this person is actually transmitted to the latter page so that it won’t show some other’s information. There are basically two means of data transmission.The first one is used to transmit single piece of data, while the other one is used to transmit a bunch of data using class *bundle*. A typical implementation is shown in Fig.8.

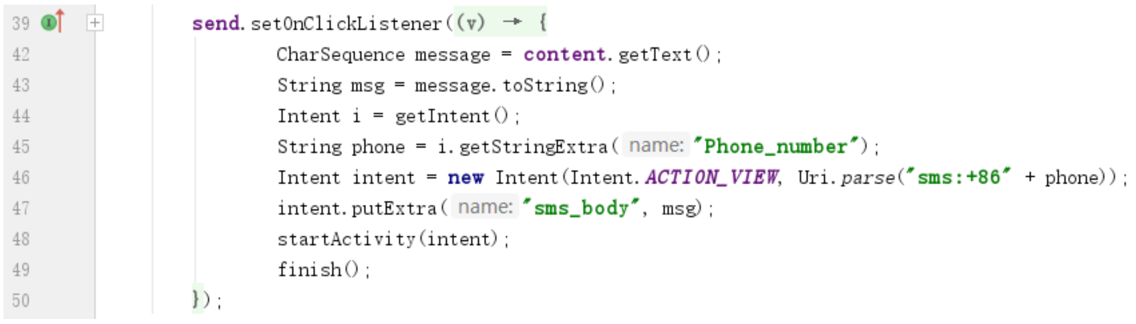


```java
Bundle bundle = new Bundle();
bundle.putString("name", name);
bundle.putInt("head", head);
bundle.putString("phone", phone);
dtl.putExtra( name: "detail_info", bundle);
startActivity(dtl);
```

Figure 8: Data transmission with bundle

3.2.4 SMS Sending

In this experiment, we put an extra function, SMS sending, to our Contact app. This function requires to call the built-in class *smsManager* aslwell as function *sendTextMessage*, or we can use *Intent.ACTION_VIEW* to pass on message content and phone number to SMS system automatically. Again, class *Intent* is used here. Different from the previous situation where *Intent* is used explicitly, here we use *Intent* implicitly. Implicit *Intent* specifies a series more abstract information like action and category, and casts the *intent* to system to analyze it and carry out appropriate activity. On a solid machine, message sending would be conducted if SIM is configured correctly. The code implementing the latter way is shown in Fig.9.



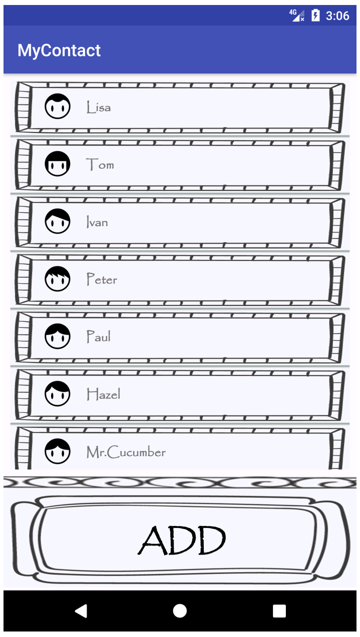
```java
send.setOnClickListener((v) -> {
        CharSequence message = content.getText();
        String msg = message.toString();
        Intent i = getIntent();
        String phone = i.getStringExtra( name: "Phone_number");
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("sms:+86" + phone));
        intent.putExtra( name: "sms_body", msg);
        startActivity(intent);
        finish();
});
```

Figure 9: SMS sending

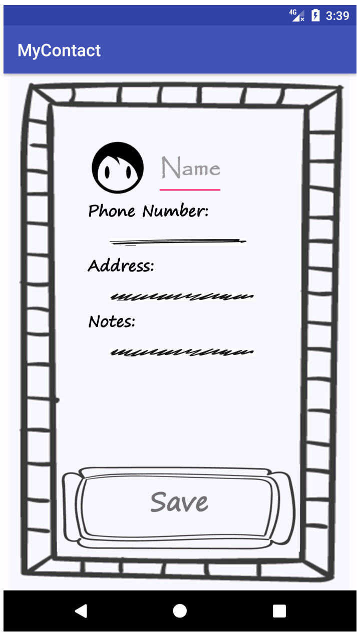
4 Experiment Result

This section shows our experiment results of App Contact using graph and necessary textual explanation. The first use case describes the process of creating a new contact and add him/her to the original contact list. The second use case illustrates the process of editing a new message to a target contact and send the message out. We present the snapshots below to better explain the functions and show our work on interfaces.

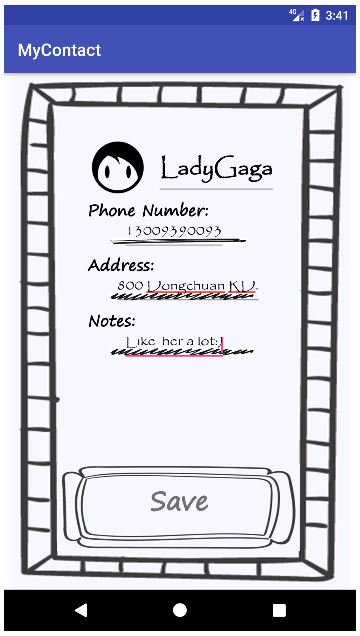
4.1 Use Case 1: Add new contacts



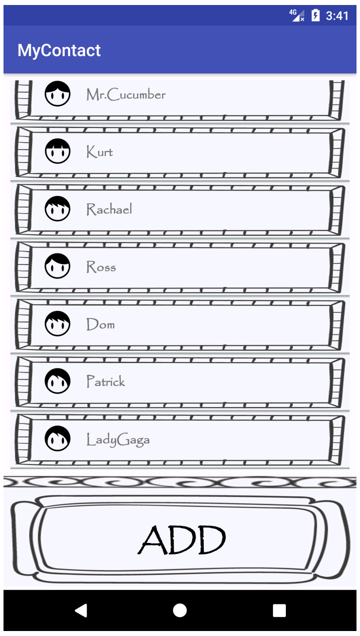
(a) Contact list



(b) Add new contact



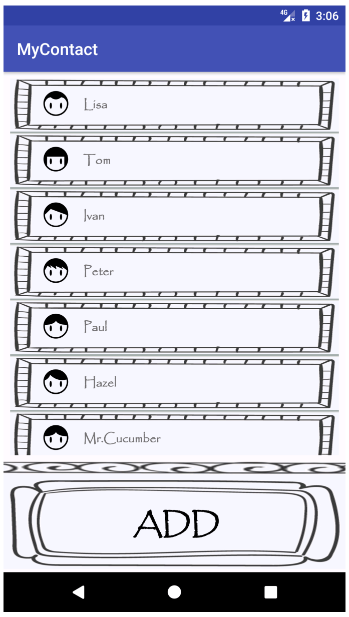
(c) That's it



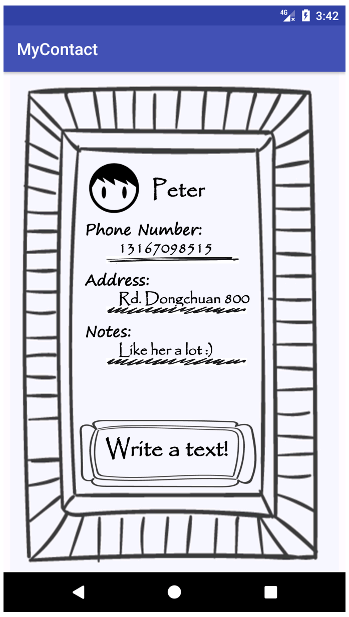
(d) Got a new one

Figure 10: Use Case 1: Add new contacts

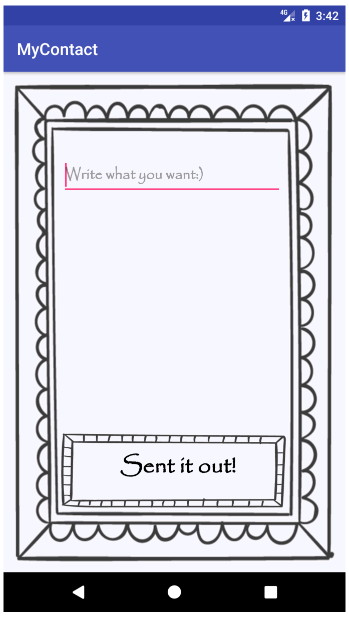
4.2 Use Case 2: Write a message



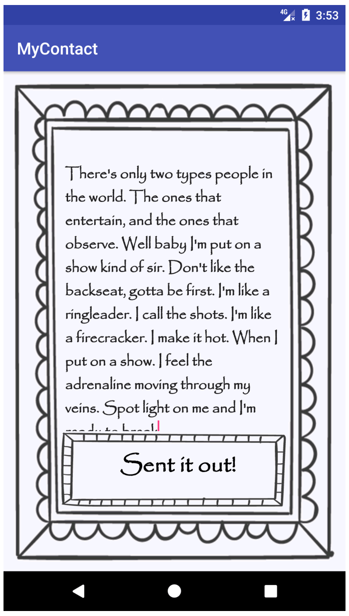
(a) Contact list



(b) Check detailed info

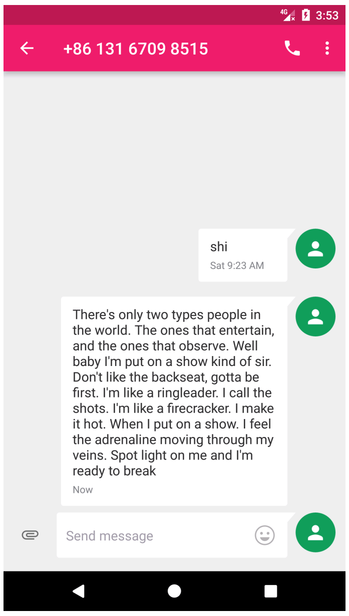


(c) Let's write to him



There's only two types people in the world. The ones that entertain, and the ones that observe. Well baby I'm put on a show kind of sir. Don't like the backseat, gotta be first. I'm like a ringleader. I call the shots. I'm like a firecracker. I make it hot. When I put on a show. I feel the adrenaline moving through my veins. Spot light on me and I'm ready to break!

(d) Finish!



There's only two types people in the world. The ones that entertain, and the ones that observe. Well baby I'm put on a show kind of sir. Don't like the backseat, gotta be first. I'm like a ringleader. I call the shots. I'm like a firecracker. I make it hot. When I put on a show. I feel the adrenaline moving through my veins. Spot light on me and I'm ready to break

(e) Done sending:)

Figure 11: Use Case 2: Write a message

5 Discussion

In this section, we propose a couple of problems we encountered in the process of implimentation and carry out further discussions on them.

5.1 Customized Adapter

Adapter is the interface connecting back-end data and front-end display. It is an important bond between data and UI (View). Fig.12 shows the relationship between data, *Adapter* and *View*.

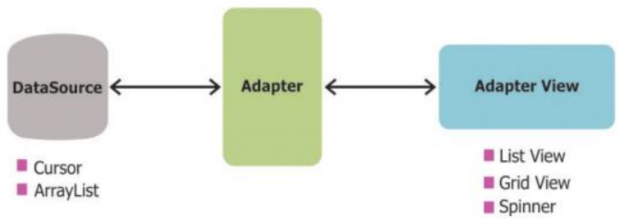
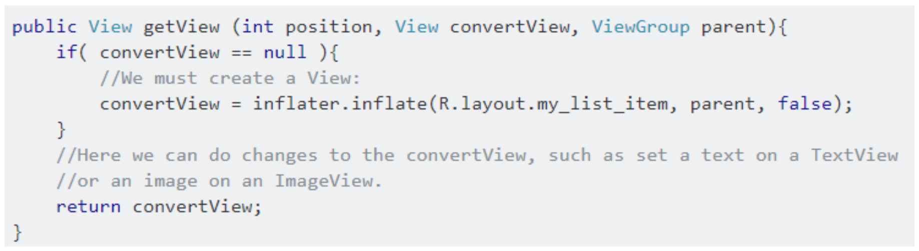


Figure 12: Threesome relationship

Built-in adapters can only handle limited classes. For instance, *ArrayAdapter* can handle lists of *String* class. However, for a customized class (denoted as *item*), *ArrayAdapter* may not work anymore. At this point, a more adaptive adapter is needed to represent data contents on screen correctly. Commonly, we write a derived class by extending proper basic *Adapter* (like *BaseAdapter*). In the new adapter class, we should introduce a *Viewholder* to store necessary attributes of every item and set value more efficiently while rolling *ListView*. Also, we should override the function *getView()* to generate *item*'s view of a *ListView* or others. A general implementation of *getVoew()* is shown in Fig.13.



```
public View getView (int position, View convertView, ViewGroup parent){
    if( convertView == null ){
        //We must create a View:
        convertView = inflater.inflate(R.layout.my_list_item, parent, false);
    }
    //Here we can do changes to the convertView, such as set a text on a TextView
    //or an image on an ImageView.
    return convertView;
}
```

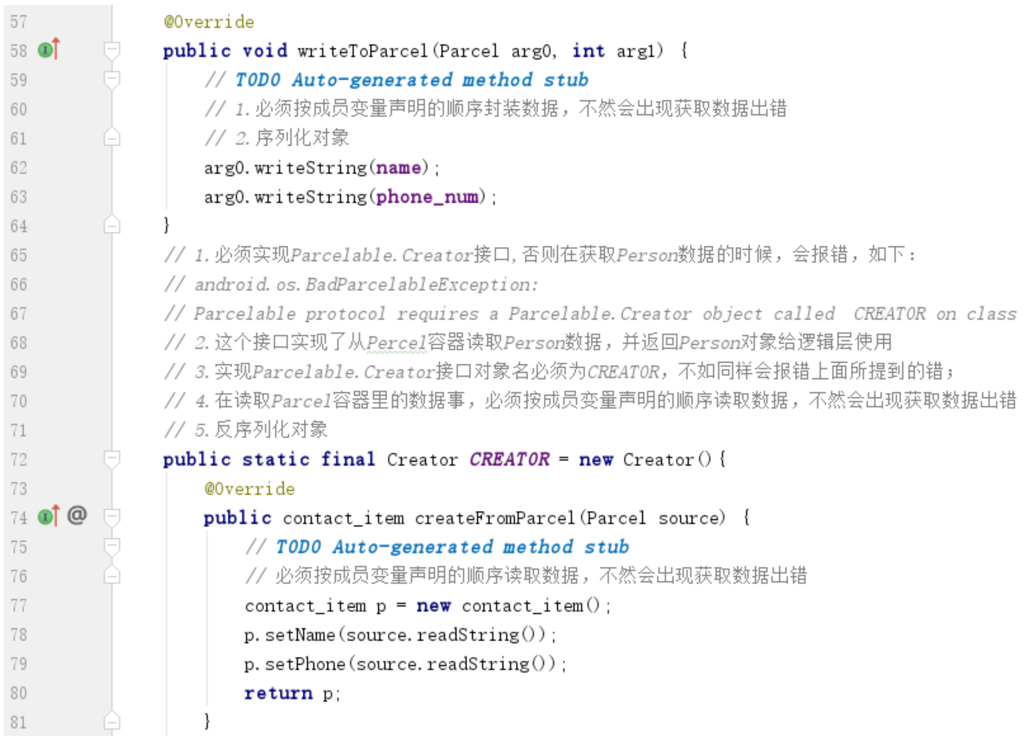
Figure 13: General getView()

5.2 Transmission of A List

We have mention that *Bundle* could be used to transmit a bunch of data instead of single piece. Howver, what if we need to transmit a list of objects? Actually, objects of simple class can still be easily transmitted even without using *Bundle*. For *String* type, the function called here should only be

$$intent.putIntegerArrayListExtra(key, list);$$

However, if the objects are customized, we should use *Serializable* means or *Parcelable* means. Correspondingly, we need to implement *Serializable* or *Parcelable* interfaces and use *Bundle* to send and receive list data. Take the latter for example, *Item* should be defined to implement *Parcelable* class. Functions *writeToParcel()*, *describeContents()* and *createFromParcel()* should be overridden. The implementation in our experiment is shown in Fig.14.



```
    @Override
    public void writeToParcel(Parcel arg0, int arg1) {
        // TODO Auto-generated method stub
        // 1.必须按成员变量声明的顺序封装数据，不然会出现获取数据出错
        // 2.序列化对象
        arg0.writeString(name);
        arg0.writeString(phone_num);
    }
    // 1.必须实现Parcelable.Creator接口,否则在获取Person数据的时候，会报错，如下：
    // android.os.BadParcelableException:
    // Parcelable protocol requires a Parcelable.Creator object called  CREATOR on class
    // 2.这个接口实现了从Percel容器读取Person数据，并返回Person对象给逻辑层使用
    // 3.实现Parcelable.Creator接口对象名必须为CREATOR，不如同样会报错上面所提到的错；
    // 4.在读取Parcel容器里的数据事，必须按成员变量声明的顺序读取数据，不然会出现获取数据出错
    // 5.反序列化对象
    public static final Creator CREATOR = new Creator(){
        @Override
        public contact_item createFromParcel(Parcel source) {
            // TODO Auto-generated method stub
            // 必须按成员变量声明的顺序读取数据，不然会出现获取数据出错
            contact_item p = new contact_item();
            p.setName(source.readString());
            p.setPhone(source.readString());
            return p;
        }
```

Figure 14: Core overridden functions