Project 2

Digital Clock and Alarm System

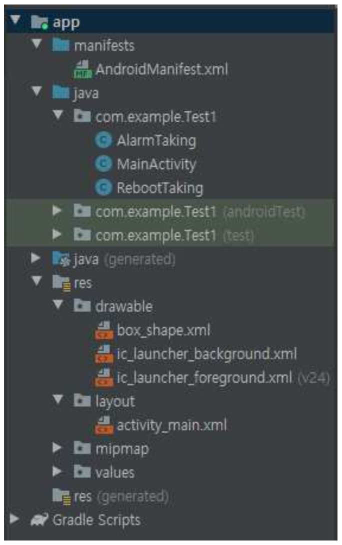
CSC 4320 Operating Systems

Hyunki Lee
Panther#: 002-34-4677

Introduction

This is a simple application that it displays current time, and a user can set alarm time. The alarm is possible to run in background. Thus, even though users can reboot their devices, the alarm will work after rebooting the devices. Alarm system is one of the most useful applications. Alarm system will remind users schedule that should be known before deadline. This application mostly is related to control threads. The current time and alarm system will be run by using threads.

Application Design



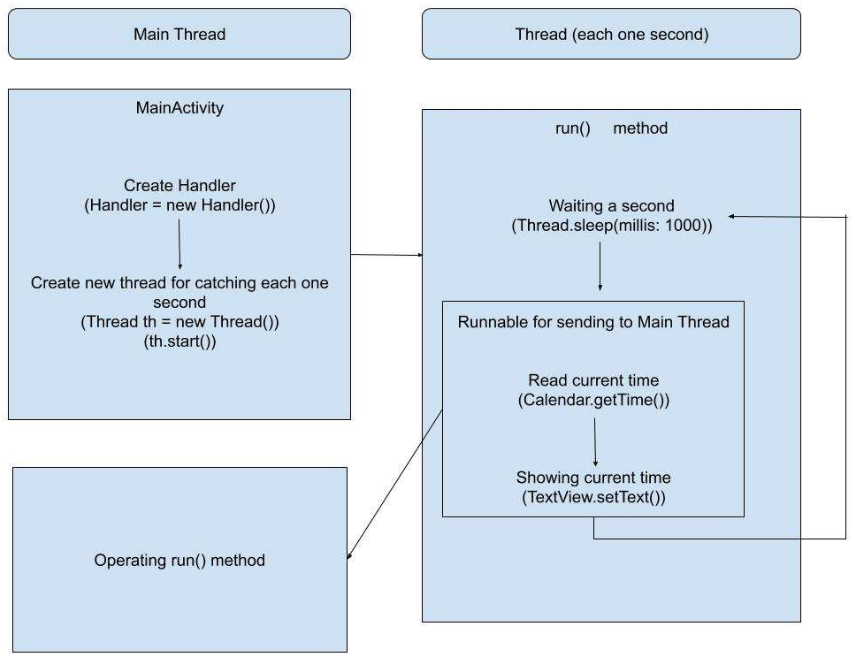
```
▼ app
    ▼ manifests
          AndroidManifest.xml
    ▼ java
        ▼ com.example.Test1
              AlarmTaking
              MainActivity
              RebootTaking
        ▶ com.example.Test1 (androidTest)
        ▶ com.example.Test1 (test)
    ▶ java (generated)
    ▼ res
        ▼ drawable
              box_shape.xml
              ic_launcher_background.xml
              ic_launcher_foreground.xml (v24)
        ▼ layout
              activity_main.xml
        ▶ mipmap
        ▶ values
      res (generated)
▶ Gradle Scripts
```

The application is built mainly five parts. The first part is AndroidManifest.xml. This part is to register necessary authority and receivers. The second part is AlarmTaking class. In this part, popup notification will show up when alarm is going off. Also, the application will open when users touch the popup message. The third part is RebootTaking class. In this class, alarm system will operate even though a device is rebooted. BroadcastReceiver is used. The forth part is MainActivity class. In this class, current time will display to use runnable and handler, and TimePicker uses to set alarm time. AlarmManager is used to get notification of alarm. The last part is Activity_Main.xml. This makes application's appearance, and it interacts between a user and an application.

In AlarmTaking class, some important APIs are Intent, BroadcastReceiver, and NotificationManager. The BroadcastReceiver is registered in AndroidManifest.xml. Thus, even if the application does not open, users can get message and take action for the message. The message will be created from NotificationManager, and BroadcastReceiver will show the message. It means the message can be running on background. Intent holds information from notificationManager. PendingIntent will take activity that is related to notification. Next step, version control condition is needed before creating channel. If the API version is higher than OREO(API 26), R.drawable keyword is used to create icon. Otherwise, R.mipmap is used. After that channel is created and using NotificationManager to get message with sounds. The created channel will be registered in the system. The popup message sets to use NotificationCompat's categories. If the notification is not null, then the application operates popup message. Using calendar, the user can get notification next day. The setting that previous information will be saved in preference. Finally, typing the contents of message that user set alarm time is needed.

In RebootTaking class, important APIs are BroadcastReceiver and AlarmManager. The AlarmManager will create systemService that if users reboot their device, the alarm system will run automatically. Also, BroadcastReceiver is registered in AndroidManifest.xml to run the application on the background. First, Intent and PendingIntent are created to hold context from AlarmTaking class and get information from Broadcast. The users need to get popup message and information of alarm time. Therefore, AlarmTaking class needs to create RebootTaking class. Second, AlarmManager API will implement to get SystemService. Alarm Manager can run application code at specific time even though the application is not currently running. We create condition that intent will act when a device reboot then we use Alarm Manager to run application. This condition is saved at SharedPreferences. SharedPreferences is useful when we need to save some simple data such as default value, checking access a certain application or not. A key will be set to get certain information from preferences. Calendar uses to set alarm schedule information. If the user reboot device and previous set alarm time is already passed, the information that in the key will update the alarm (next day). The popup message and notification for alarm time will display to use date_text and toast. Furthermore, we need to create condition that if the AlarmManager has information, the system will repeat the RebootTaking class codes each day.

In MainActivity class, there are two parts which are displaying current time and alarm system. For displaying current time, Handler is an important API. Handler is a tool for communication with threads. Typically, handler is used with message and looper. Message is an object that has data and identifier. The messages are in message queue. The looper takes a message from message queue then sends to handler. In this application, runnable is used instead of message since the runnable object also can be sent to handler. The main purpose that using message and handler to communicate with threads is that data will be sent from handler and then run the threads code. This is not a simple way to use threads. We need to set an identifier to distinguish the types of data in message object. In handlerMessage() method, we need to create condition codes for the identifier. The diverse messages need depend on the types of data. If our purpose is to use handler for running thread's code, we can just send 'running code' to threads. It means that we send an object that have running code to handler, and threads receive the running code from handler. The threads directly run the code that it receives. The object that have running code is 'runnable' object. The two threads will be used. One thread is for receiving, and the other one is for sending. In the thread (receiving), handler object will be created. When handler is created in the thread, the handler will automatically connect to the thread's looper and MessageQueue. In the thread (sending), runnable object and run() method (overriding) are created. The runnable object will send to handler.post() method. Runnable instance will be created in run() method (override). The runnable interface only has run() method. The follow diagram is representing the structure of this applications' digital clock.

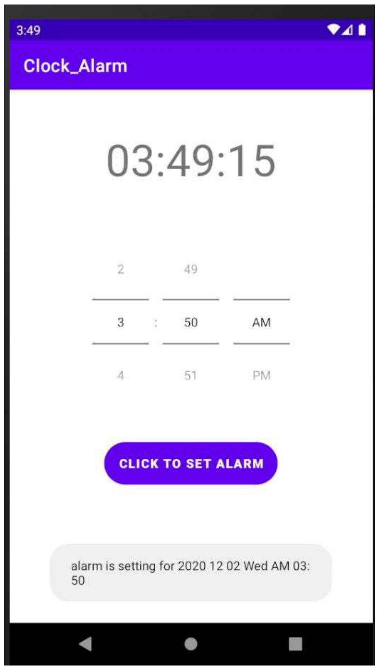


In actual code, first handler is created, and runnable object is created. Run() method is created in the runnable object (override). In the run() method, calendar is created to get instance also import the time format (hours : minutes : seconds) by using SimpleDataFormat. A string variable is created to save the time. The actual time comes from calendar, and the format of time comes form SimpleDataFormat. The String displays from clockText method. Next, new thread is created to work for waiting time. In the new thread, thread.sleep method is used to calculate each one second and then using handler.post() method to send the calculated time to main thread. There are some important contents. First, some works that operate concurrently with main thread should be implemented in other threads. Second, In main thread, we should avoid tasks that take long time for running or waiting. Third, a task that changes user interface should be running in main thread. Finally, we can use handler to send messages from handler to main thread.

The second part of MainActivity class is alarm system part. TimePicker and AlarmManager are the main APIs. TimePicker is for setting alarm time, and AlarmManager will go off alarm at the proper time. In the actual code, TimePicker is created first, and we can decide the format of time's view. For example, the method picker.setIs24Hourview() can decide the format. If we type 'true', the alarm setting view will represent time that starts from 0 ~ 24 hours. Otherwise, If we type 'false' the alarm setting view will show separated time format such as from 0 ~ 12 AM, from 0 ~ 12PM. SharedPreferences is used to set default value of time. The default value is current time if the alarm time is not setting. Calendar is used to set next notification time. We use SimpleDateFormat to make message format that users will see. Toast is used to send the message to users. Next step is to initialize TimePicker setting as previous setting. For example, this application can set only one alarm, thus if the alarm time past, then the next alarm time should be the same time at next day. Unless, users change the alarm time, the set time will continue. Time format is string, thus parseInt method is used to save the time information as integer. After that depending on API level, we have to use different method. If the API lever is higher or equal to 23, we use picker.setHour and picker.setMinute to initialize TimePicker. Otherwise, If the API level is lower than 23, we need to use picker.setCurrentHour and picker.setCurrentMinute to initialize TimePicker. A button is created, and button.setOnClickListener() method is created to decide acting. In the method, the condition (Depending on API level) is created then separate time that is AM and PM in case using AM, PM format. The actual setting alarm function is in calendar method. Calendar method can set alarm time that users decide. We need to use calendar.set method for each unit (hours, minutes, seconds). The next if statement is for next alarm setting. If the alarm time that is set is past, the calendar directly will add new alarm date that next day. In this case, calendar.add() method is used. After update the calendar, message will toast to users. The setting that mentioned above will be saved in preference. The last function is diary notification. In this function, if users approve daily alarm, the alarm will show up every day. An intent will store AlarmTaking class then pending intent will send the information to BroadCast. If users approve the daily alarm, the alarmManager() method will repeat the alarm. AlarmManager.setRepeating() method is used. Finally, a code makes possible to use receiver after rebooting a device. For this code, we need activity. Activity is the one foundation unit to consist of UI. Usually, the activity is controlled in Manifest.xml, but we can control activity in code. PackageManager.setComponentEnabledSetting() method is the way to control activity. If we set COMPONENT_ENABLED_STATE_ENABLED, the inactivated activity becomes activated activity. Furthermore, if we set DON'T_KILL_APP, the application will not turn off. Thus, even if users turn off this application or reboot their devices, the activity will continue.

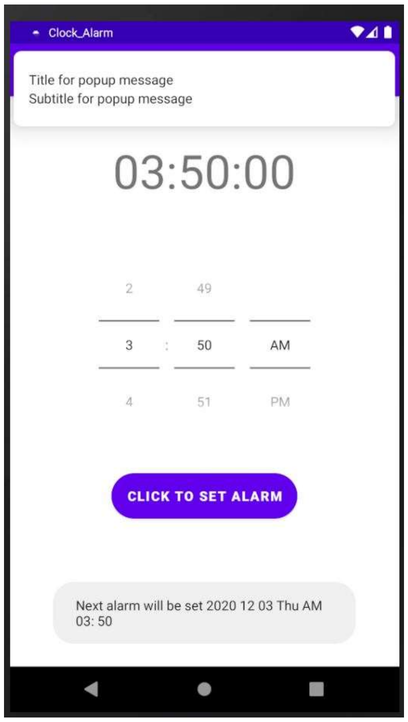
In activity_main.xml, codes are to design the appearance of the application. Also, button and timePicker is designed to interact between users and the application. The TimePicker connects with TimePicker in MainActivity class. The user can decide the alarm time to see the TimePicker UI. In the Button case, the UI is connecting with Button in MainActivity class. If users click the button in their device, the system will set the alarm time that users pick. We need to use layout_constraint() method to place each UI's position.

User Interface is following screenshots.



alarm is setting for 2020 12 02 Wed AM 03: 50

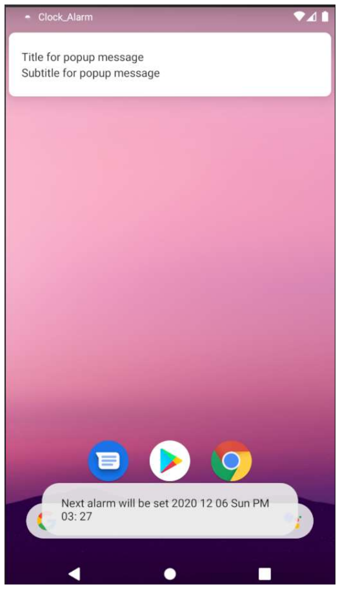
The application name shows the left-top of the application. The bottom application name displays current time. In this screenshot, we can see the current time and the displayed time in application correspond each other. The time picker is placed in the middle of the application. Users can scroll up and scroll down to find time that they want. Also, user can choose AM or PM. If users scroll up or down from some time to over 12 (in hours unit), the AM, PM will be changed automatically. Under the time picker, users can see a button to set alarm. If users click the button, users will see a message under the button. The message shows that when the alarm will go off. The first screenshot shows the next alarm will be on 2020-12-02, Wed, 3:50 am.



Title for popup message
Subtitle for popup message

Next alarm will be set 2020 12 03 Thu AM 03: 50

In the second screenshot, when the time is up, users can see the popup message on the top of the application. Also, users can see a message from the bottom of the button. The message shows the next alarm time. The first alarm time which is 2020-12-02, 3:50 am is past, thus the next alarm is set as tomorrow. (2020-12-03, Thu, 3:50 am).



Title for popup message
Subtitle for popup message

Next alarm will be set 2020 12 06 Sun PM 03: 27

The last screenshot shows that if users set alarm time, the alarm system will run background. As a result, even if users close the application, users will see the popup message and notification for next time alarm.

Source code

AlarmTaking.java

```
// need to add channel for higher than api 26
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {

    builder.setSmallIcon(R.drawable.ic_launcher_foreground);


    String channelName ="Channel for daily alarm";
    String description = "Alarming when time is set";
    int importance = NotificationManager.IMPORTANCE_HIGH; //showing sound and
alarm message

    NotificationChannel channel = new NotificationChannel("default", channelName,
importance);
    channel.setDescription(description);

    if (notificationManager != null) {
        // applying notification system to system
        notificationManager.createNotificationChannel(channel);
    }
}else builder.setSmallIcon(R.mipmap.ic_launcher); // for lower than OREO.
```

```java
if (notificationManager != null) {

    // operating notification
    notificationManager.notify(1234, builder.build());

    Calendar nextNotifyTime = Calendar.getInstance();

    // overlap alarm time for next day
    nextNotifyTime.add(Calendar.DATE, 1);

    //  saving a setting info to preference
    SharedPreferences.Editor editor = context.getSharedPreferences("daily alarm",
MODE_PRIVATE).edit();
    editor.putLong("nextNotifyTime", nextNotifyTime.getTimeInMillis());
    editor.apply();

    Date currentDateTime = nextNotifyTime.getTime();
    String date_text = new SimpleDateFormat("yyyy MM dd EE a hh: mm",
Locale.getDefault()).format(currentDateTime);
    Toast.makeText(context.getApplicationContext(),"Next alarm will be set " +
date_text, Toast.LENGTH_LONG).show();
}
```

RebootTakning.java

```java
public class RebootTaking extends BroadcastReceiver {
    @RequiresApi(api = Build.VERSION_CODES.KITKAT)
    @Override
    public void onReceive(Context context, Intent intent) {
        if (Objects.equals(intent.getAction(),
"android.intent.action.BOOT_COMPLETED")) {

            // after booting device, reset an alarm
            Intent alarmIntent = new Intent(context, AlarmTaking.class);
            PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0,
alarmIntent, 0);

            AlarmManager manager = (AlarmManager)
context.getSystemService(Context.ALARM_SERVICE);


            SharedPreferences sharedPreferences =
context.getSharedPreferences("daily alarm", MODE_PRIVATE);
            long millis = sharedPreferences.getLong("nextNotifyTime",
Calendar.getInstance().getTimeInMillis());


            Calendar current_calendar = Calendar.getInstance();
            Calendar nextNotifyTime = new GregorianCalendar();

nextNotifyTime.setTimeInMillis(sharedPreferences.getLong("nextNotifyTime",
millis));

            if (current_calendar.after(nextNotifyTime)) {
                nextNotifyTime.add(Calendar.DATE, 1);
            }

            Date currentDateTime = nextNotifyTime.getTime();
            String date_text = new SimpleDateFormat("yyyy MM dd EE a hh: mm",
Locale.getDefault()).format(currentDateTime);
            Toast.makeText(context.getApplicationContext(),"[Reboot] Next alarm
will be set " + date_text, Toast.LENGTH_LONG).show();


            if (manager != null) {
                manager.setRepeating(AlarmManager.RTC_WAKEUP,
nextNotifyTime.getTimeInMillis(),
                        AlarmManager.INTERVAL_DAY, pendingIntent);
            }
        }

    }
}
```

MainActivity.java

```
//using handler and thread(run())
cHandler = new Handler();
final Runnable runna = new Runnable() {
    @Override
    public void run() {
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        String str = sdf.format(cal.getTime());
        clockText = findViewById(R.id.clock);
        clockText.setText(str);
    }

};

class NewRunnable implements Runnable {

    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(1000); // 1 sec
            } catch (Exception e) {
                e.printStackTrace();
            }
            cHandler.post(runna);
        }
    }
}
NewRunnable nrun = new NewRunnable();
Thread th = new Thread(nrun);
th.start();
```

```
// Default is current time
// If the time is set for alarm. show the previous setting time
SharedPreferences sharedPreferences = getSharedPreferences("daily alarm",
MODE_PRIVATE);
long millis = sharedPreferences.getLong("nextNotifyTime",
Calendar.getInstance().getTimeInMillis());
```

```java
Calendar nextNotifyTime = new GregorianCalendar();
nextNotifyTime.setTimeInMillis(millis);

Date nextDate = nextNotifyTime.getTime();
String date_text = new SimpleDateFormat("yyyy MM dd EE a hh: mm",
Locale.getDefault()).format(nextDate);
Toast.makeText(getApplicationContext()," Next alarm will be " + date_text,
Toast.LENGTH_LONG).show();


// using previous setting initializing TimePicker
Date currentTime = nextNotifyTime.getTime();
SimpleDateFormat HourFormat = new SimpleDateFormat("kk", Locale.getDefault());
SimpleDateFormat MinuteFormat = new SimpleDateFormat("mm", Locale.getDefault());

int pre_hour = Integer.parseInt(HourFormat.format(currentTime));
if(pre_hour == 24)
    pre_hour = 0; //timePicker.setHour(0~23)
int pre_minute = Integer.parseInt(MinuteFormat.format(currentTime));

//Defend on api level.  min 16 for current setting
if (Build.VERSION.SDK_INT >= 23 ){
    picker.setHour(pre_hour);
    picker.setMinute(pre_minute);
}
else{
    picker.setCurrentHour(pre_hour);
    picker.setCurrentMinute(pre_minute);
}
```

```java
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {

        int hour, hour_24, minute;
        String am_pm;
        if (Build.VERSION.SDK_INT >= 23 ){
            hour_24 = picker.getHour();
            minute = picker.getMinute();
        }
        else{
            hour_24 = picker.getCurrentHour();
            minute = picker.getCurrentMinute();
        }
        if(hour_24 > 12) {
            am_pm = "PM";
            hour = hour_24 - 12;
        }
        else
        {
            hour = hour_24;
            am_pm="AM";
        }

        // set a alarm
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(System.currentTimeMillis());
        calendar.set(Calendar.HOUR_OF_DAY, hour_24);
        calendar.set(Calendar.MINUTE, minute);
        calendar.set(Calendar.SECOND, 0);

        // If the alarm time is past, automatically set the same time for next day
        if (calendar.before(Calendar.getInstance())) {
            calendar.add(Calendar.DATE, 1);
        }

        Date currentDateTime = calendar.getTime();
        String date_text = new SimpleDateFormat("yyyy MM dd EE a hh: mm",
Locale.getDefault()).format(currentDateTime);
        Toast.makeText(getApplicationContext(), "alarm is setting for " +
date_text , Toast.LENGTH_LONG).show();

        //  Saving a value that is set in preferences
        SharedPreferences.Editor editor = getSharedPreferences("daily alarm",
MODE_PRIVATE).edit();
        editor.putLong("nextNotifyTime", (long)calendar.getTimeInMillis());
        editor.apply();
```

```java
void diaryNotification(Calendar calendar)
{

    Boolean dailyNotify = true; // must use alarm system

    PackageManager pm = this.getPackageManager();
    ComponentName receiver = new ComponentName(this, RebootTaking.class);
    Intent alarmIntent = new Intent(this, AlarmTaking.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, alarmIntent,
0);
    AlarmManager alarmManager = (AlarmManager)
getSystemService(Context.ALARM_SERVICE);


    // if user can approve daily alarm ...
    if (dailyNotify) {


        if (alarmManager != null) {

            alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(),
                    AlarmManager.INTERVAL_DAY, pendingIntent);

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                alarmManager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(), pendingIntent);
            }
        }

        // after booting system, still the receiver is working
        pm.setComponentEnabledSetting(receiver,
                PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
                PackageManager.DONT_KILL_APP);

    }
```