Department of Electronic Engineering

City University of Hong Kong

EE5415 Mobile Applications Design and Development

Lab06: Intents and Activity Lifecycle (Java)

I. Inter-activity Communication

In general, mobile apps use dialogs and screens to present data to or otherwise interact with the user. Typically, a dialog or screen is used for a single concept or operation in mobile app. In Android, such concept or operation with an associated screen is called an **activity**.

An Android app has one or more activities, in which one of them is the default activity. This default activity is started when the app is launched. The activities and the default activity of an Android app are declared in the manifest file (`AndroidManifest.xml`).

In addition, communication among Android activities is achieved using intents to promote reuse and consistence in user experience. **An intent is essentially an intention, or a description of a desired operation to be performed**. Intent is represented by the `android.content.Intent` class. A common use of intents is starting another activity within the same app. This can be achieved by invoking the `startActivity(Intent)` method of the Activity class, as codes listed in the following block. In this lab, students will experience how an activity starts another activity using **explicit** and **implicit** intents for passing of data between activities or applications.

1.1 Explicit and Implicit Intents

An **intent** is essentially an intention, a description of a desired operation to be performed. An intent can be either explicit or implicit:

- An ***explicit intent*** specifies the class of an activity or a component to be started. This may be used to start an internal activity from the same app.
- An ***implicit intent*** specifies a set of information, or constraints, that the system uses to determine a suitable activity or component to be started. The mechanism of implicit intents is very flexible as the system can select the best way to handle an intent. An example of an implicit intent is viewing a particular webpage; the intent would consist of the action of viewing and the data of the webpage URL. To respond to such an intent, the Android system typically starts a Web browser and loads the page.

An intent is represented by the `android.content.Intent` class. Regarding the information of an operation to be performed, an intent contains one or more of the following items:

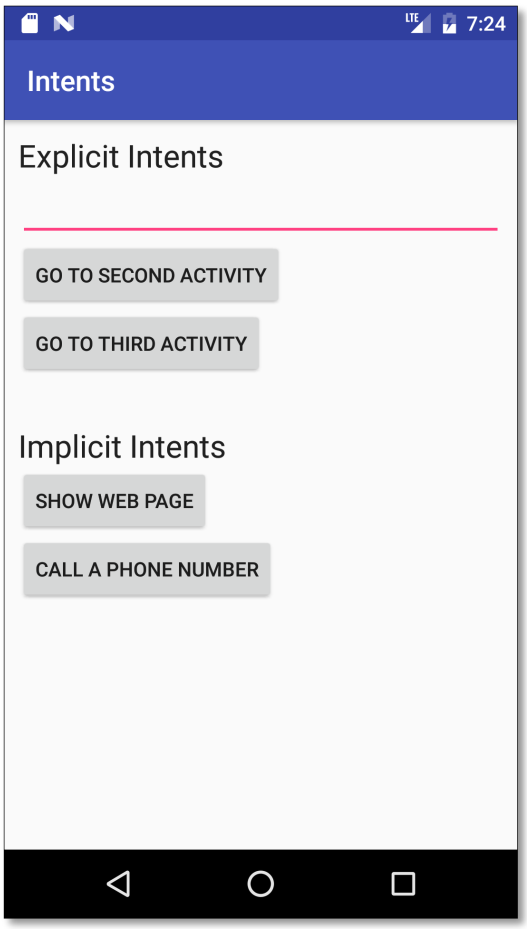
- ***Component*** – name of the activity or component to be started
- ***Action*** – action to be performed; some constants are defined in the `Intent` class for various actions, e.g., `ACTION_VIEW` for viewing and `ACTION_CALL` for making a call
- ***Data*** – data to be acted on in form of a URI, and the MIME type of the data, e.g., the URI of 'http://www.google.com'
- ***Type*** – an explicit type that overrides the MIME type of the data
- ***Category*** – additional information about the action, e.g., the constant `CATEGORY_LAUNCHER` of the `Intent` class specifying an activity as a top-level application that appears in the launcher
- ***Extras*** – key-value pairs of additional information to be passed to the activity or component to be started, e.g., the subject and body of a message for an activity to send emails
- ***Flags*** – flags for controlling how the system launches or otherwise handle the activity or component to be started

To build an intent, not all these items are required. In practice, we often use the explicit component name or the implicit combination of an action and data to start an activity.

There are various constructors and methods of the `Intent` class for creating explicit and implicit intents. With a properly constructed intent, an activity starts another activity by invoking the `startActivity(Intent)` method of the `Activity` class.

1.2 Using Explicit and Implicit Intents

In this section, students are required to build an example "**Intents**" app for demonstrating practical uses of both explicit and implicit intents. The app consists of four buttons with two buttons to call methods for creating explicit intents using `startActivity()` and `startActivityForResult()` methods to start another activities, and the other two buttons to call methods using implicit intents to open a webpage and call a phone number. In these implicit intents, the Android intent resolution algorithm will be engaged to identify and launch a suitable activity from another application for showing the webpage or dialing a phone call. In the case of opening a webpage, this is most likely to be an activity from the Chrome web browser bundled with the Android operating system. The user interface of the example "Intents" app is as shown below:



1.2.1 Creating an Intents Example Project with App Name of "Intents"

Within the Android Studio environment, create a new Android Application project with project name of "**Intents**", appropriate package name and SDK selections.

1.2.2 Build the User Interface

The user interface for the `MainActivity` class is very simple, which consisting solely of a `LinearLayout` view with two `TextView` and four buttons. Within Android Studio Project view window, locate the **`res -> layout -> activity_main.xml`** file and double click on it to load it into the editing pane. Select the `activity_main.xml` tab at the bottom of the editing area and replace the current XML with the following:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="1"
    android:padding="10dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal|left"
        android:text="Explicit Intents"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="goToSecondActivity"
        android:text="Go to second activity" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="goToThirdActivity"
        android:text="Go to third activity" />

    <Space
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.17" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal|left"
        android:text="Implicit Intents"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button_webpage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showWebPage"
        android:text="Show Web Page" />

    <Button
        android:id="@+id/button_call"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="callNumber"
        android:text="Call a Phone Number" />

</LinearLayout>
```

The first explicit intent is designed to start another activity called `SecondActivity`, using the wizard (right **click "com.example.intents" -> New -> Activity -> Empty Activity)** to create a blank activity with name of `SecondActivity` and Layout Name of `activity_second`. Thus, the user interface for this second activity is located in `res -> layout -> activity_second.xml` file, in which, there are one `TextView` and an `EditText` for display the title of the activity and the message from the `MainActivity` by intent. The XML code of the `activity_second.xml` file is listed below:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Second Activity"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
</LinearLayout>
```

The second explicit intent is designed to start another activity called `ThirdActivity`, using the wizard (right **click "com.example.intents" -> New -> Activity -> Empty Activity)** to create a blank activity with name of `ThirdActivity` and Layout Name of `activity_third`. Then, the user interface for this third activity is located in `res -> layout -> activity_third.xml` file, in which, there are one `TextView`, an `EditText` and a button for display the title of the activity, input message pass back to the `MainActivity` by intent and a back button. The XML code of the `activity_third.xml` file is listed below:

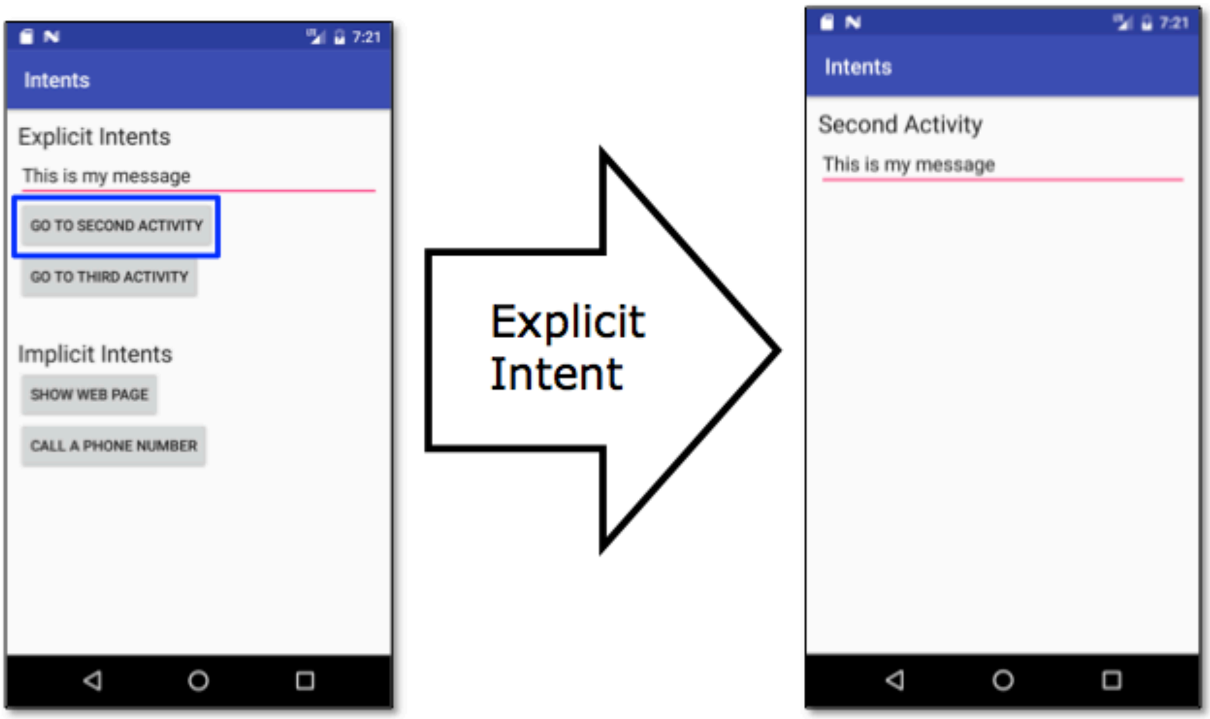
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Third Activity"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Back to Main Activity"
        android:onClick="backButton"
        />
</LinearLayout>
```

1.2.3 Creating the Explicit Intents and Implicit Intents

As mentioned above, the two explicit intents will be created to start `SecondActivity` and `ThirdActivity` activities using `StartActivity()` and `StartActivityForResult()` methods. While there are two implicit intents for showing webpage and making a phone call. Edit the MainActivity.java Class file with the code for implementing these intents as listed below.

```
public class MainActivity extends AppCompatActivity {

    EditText et;
    private static final int REQUEST_CODE = 1;
    private static final int REQUEST_CODE_PERMISSION = 2;

    private static String[] PERMISSIONS_REQ = {
            Manifest.permission.CALL_PHONE,
    };


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et = (EditText) findViewById(R.id.editText1);

        // Checking and Seeking for CALL_PHONE permission
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission. CALL_PHONE}, REQUEST_CODE);
        }
    }

    public void goToSecondActivity(View v) {

        String myMessage = et.getText().toString();

        Intent intent = new Intent(this, SecondActivity.class);
        Bundle bundle = new Bundle();
        bundle.putString("Message", myMessage);
        intent.putExtras(bundle);
        startActivity(intent);
    }

    public void goToThirdActivity(View view) {

        Intent intent = new Intent(this, ThirdActivity.class);
        startActivityForResult(intent, REQUEST_CODE);
    }

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

        if (requestCode == REQUEST_CODE) {
            if (resultCode == RESULT_OK) {
                et.setText(data.getData().toString());
            }
        }
    }

    public void showWebPage(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://developer.android.com"));
        startActivity(intent);
    }
    public void callNumber(View view) {
        Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:34426789"));
        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
            return;
        }
        startActivity(intent);
    }

}
```

In this `MainActivity`, the `goToSecondAcitivty()` method is used to implement the explicit intent for starting the `SecondActivity`. A string message with variable name of `myMessage` is obtained from the `EditText` in the `activity_main.xml`, which is embedded into the intent as a `Bundle` object by the `intent.putExtras()` method. In the `SecondActivity`, this string message will be display in an `EditText` view as shown below.

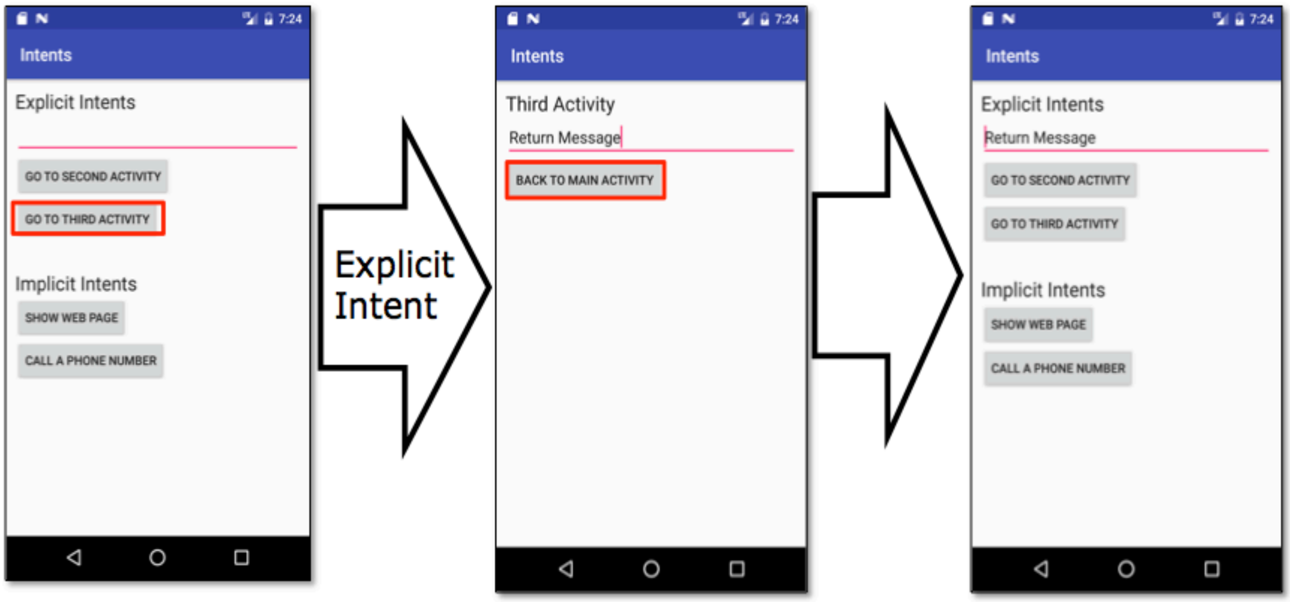


To achieve this, the `SecondActivity.java` Class file has to include the following source code for extracting the message from the intent:

```
public class SecondActivity extends AppCompatActivity {

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

        Bundle bundle = getIntent().getExtras();
        String receivedMessage = bundle.getString("Message");

        EditText myMessage = (EditText) findViewById(R.id.editText);
        myMessage.setText(receivedMessage);
    }
}
```

In the SecondActivity, the string message is obtained from the intent using the `getIntent().getExtras()` method and then pass it to a String variable of `myMessage` for displaying it in the `myMessage EditText` object using the `myMessage.setText()` method. Such that the text message entered in the `MainActivity EditText` box could be displayed in this activity.

On the other hand, the `MainActivity` also has a `goToThirdAcitivty()` method that is used to demonstrate the use of `StartActivityForResult()` method for the explicit intent with return message from the `ThirdActivity` as shown below:

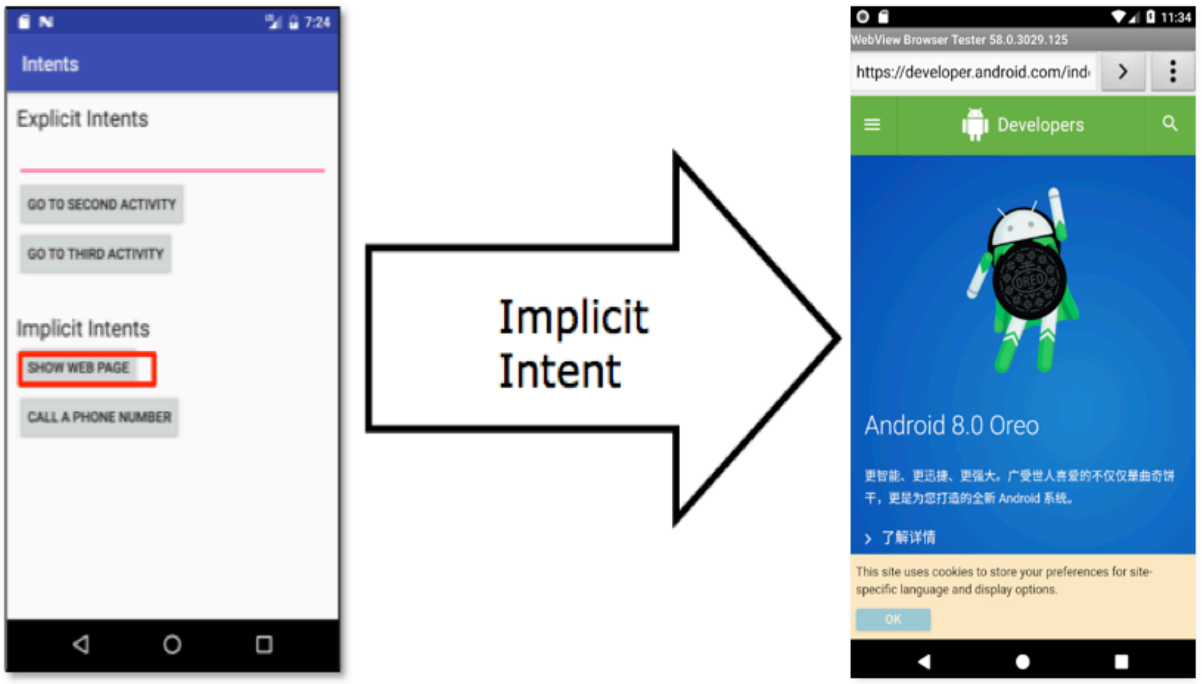


In the `ThirdActivity`, a string message with variable name of `returnMessage` is obtained from the `EditText` in the `activity_third.xml`, which is embedded into the intent using the `data.setData()` method for passing back to the `MainActivity`. This return message will be displayed in `EditText` view of the `MainActivity`. To achieve this, the `ThirdActivity.java` Class file has to include the following source code:

```
public class ThirdActivity extends AppCompatActivity {

    int REQUEST_CODE = 1;

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

    public void backButton(View view) {
        Intent data = new Intent();

        EditText returnMessage = (EditText) findViewById(R.id.editText);
        data.setData(Uri.parse(returnMessage.getText().toString()));
        setResult(RESULT_OK, data);

        finish();
    }
}
```

1.2.4 Using Implicit Intents

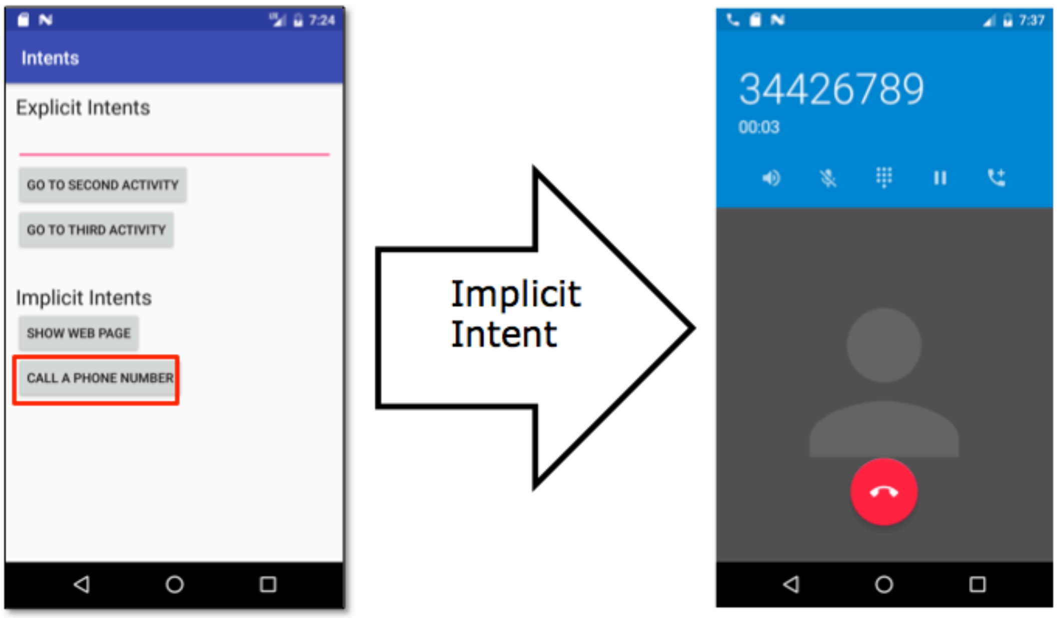
Let's turn to consider implicit intents. A common way of creating an implicit intent is by using an action and a URI data. The two examples of using implicit intents are shown in the `MainActivity.java` class file within the methods of `showWebPage()` and `callNumber()` for invoking the Web browser and making a call, respectively. In the `showWebPage()` method, an `android.net.Uri` object for the Android Developer website is created, using the static method `Uri.parse` to parse a string to create it. Then, an intent is constructed with a viewing action, which is denoted by the constant `Intent.ACTION_VIEW`, and the `Uri` object. Calling the `startActivity` method on the intent results in starting a Web browser as shown below.



更智能、更迅捷、更强大。广受世人喜爱的不仅仅是曲奇饼干，更是为您打造的全新 Android 系统。

This site uses cookies to store your preferences for site-specific language and display options.

On the other hand, the intent in the `callNumber` method specifies a calling action of `Intent.ACTION_CALL` and the telephone number of 3442-6789. It starts phone-calling activity to actually make the phone call as shown below.



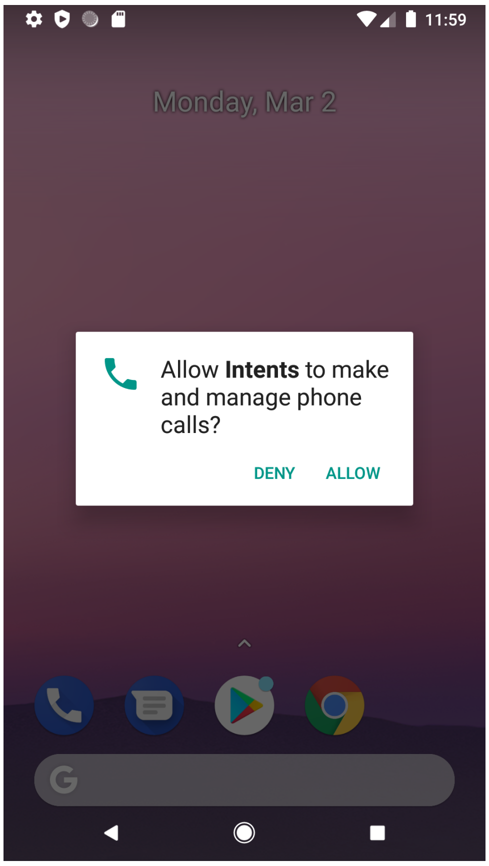
Notice that the calling action requires the permission of `android.permission.CALL_PHONE`. This permission request is specified using a `uses-permission` tag in the application's manifest file. Add the following code into the **`AndroidManifest.xml`**

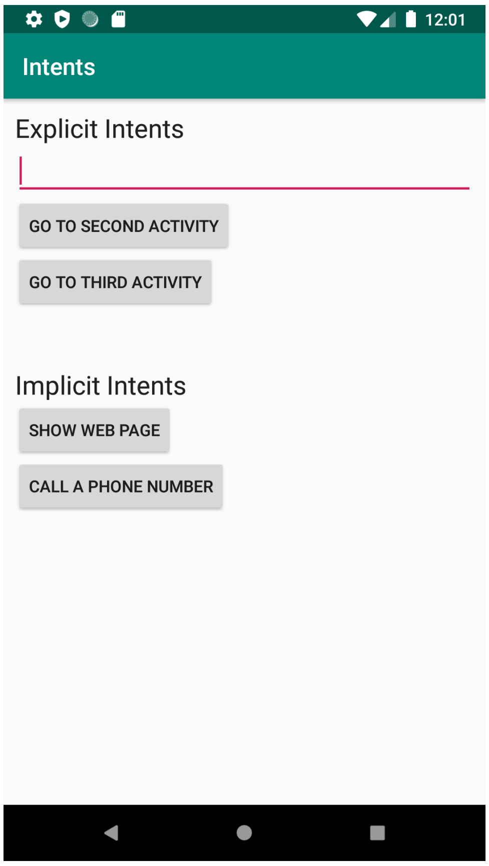
```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

In addition, we have also added the code to check this permission in the `onCreate()` function of the `MainActivity` class.

```
// Checking and Seeking for CALL_PHONE permission
if (ContextCompat.checkSelfPermission(this, Manifest.permission.CALL_PHONE) !=
PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.
CALL_PHONE}, REQUEST_CODE);
}
```

When, this Intents app is first executed, the Android system will promote for seeking allowance on use the PHONE CALL permission as shown in the following figures. We just need to press the "ALLOW", then the app will run with this permission.





Check Point 1:

1. Show your completed "Intents" app.
Instructor Verification (separate page)

II. Managing the Activity Lifecycle

As a user navigates through, out of, and back to your app, the `Activity` instances in your app transition between different states in their lifecycle. For instance, when your activity starts for the first time, it comes to the foreground of the system and receives user focus. During this process, the Android system calls a series of lifecycle methods on the activity in which you set up the user interface and other components. If the user performs an action that starts another activity or switches to another app, the system calls another set of lifecycle methods on your activity as it moves into the background (where the activity is no longer visible, but the instance and its state remains intact).

Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity. For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app. When the user returns, you can reconnect to the network and allow the user to resume the video from the same spot.

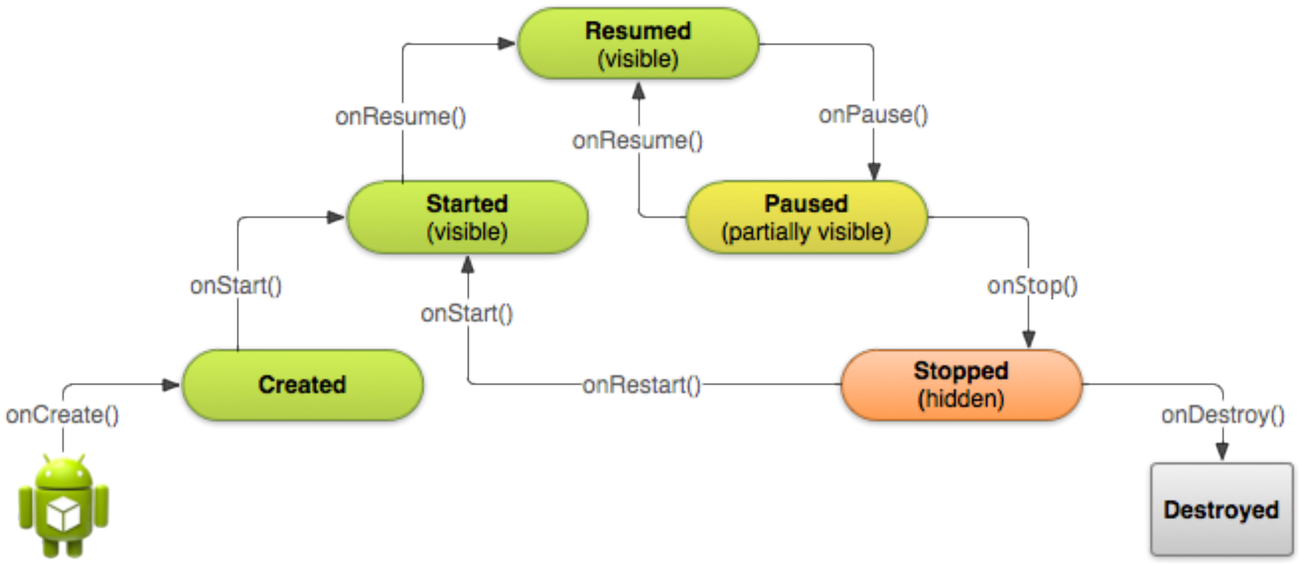
Go to the Android Developer Training class for “Managing the Activity Lifecycle” as listed below and download the demo app for studying the Android activity lifecycle.

Training class: http://developer.android.com/training/basics/activity-lifecycle/index.html

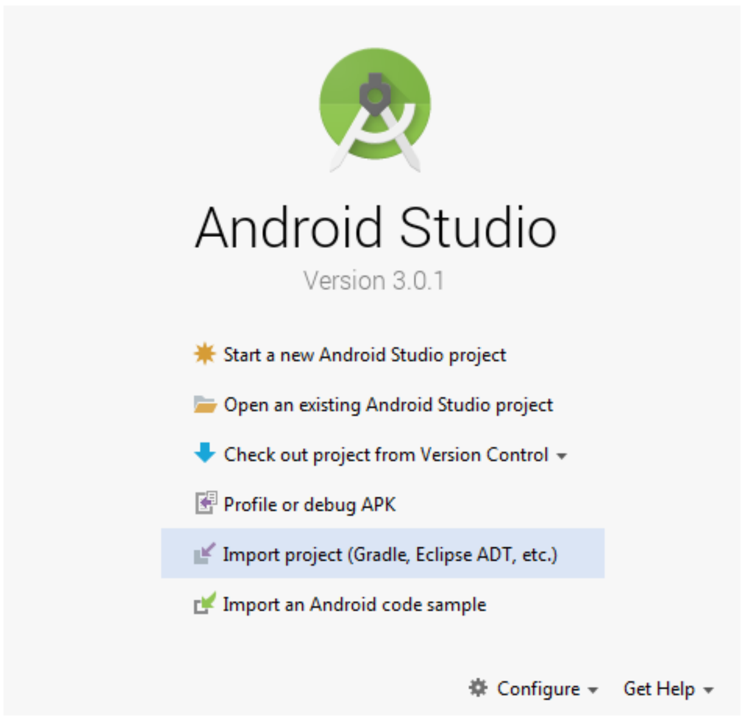
Demo App Download: http://developer.android.com/shareables/training/ActivityLifecycle.zip

This class explains important lifecycle callback methods that each `Activity` instance receives and how you can use them so your activity does what the user expects and does not consume system resources when your activity doesn't need them.

The following figure shows a simplified illustration of the Activity lifecycle, expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there's a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.

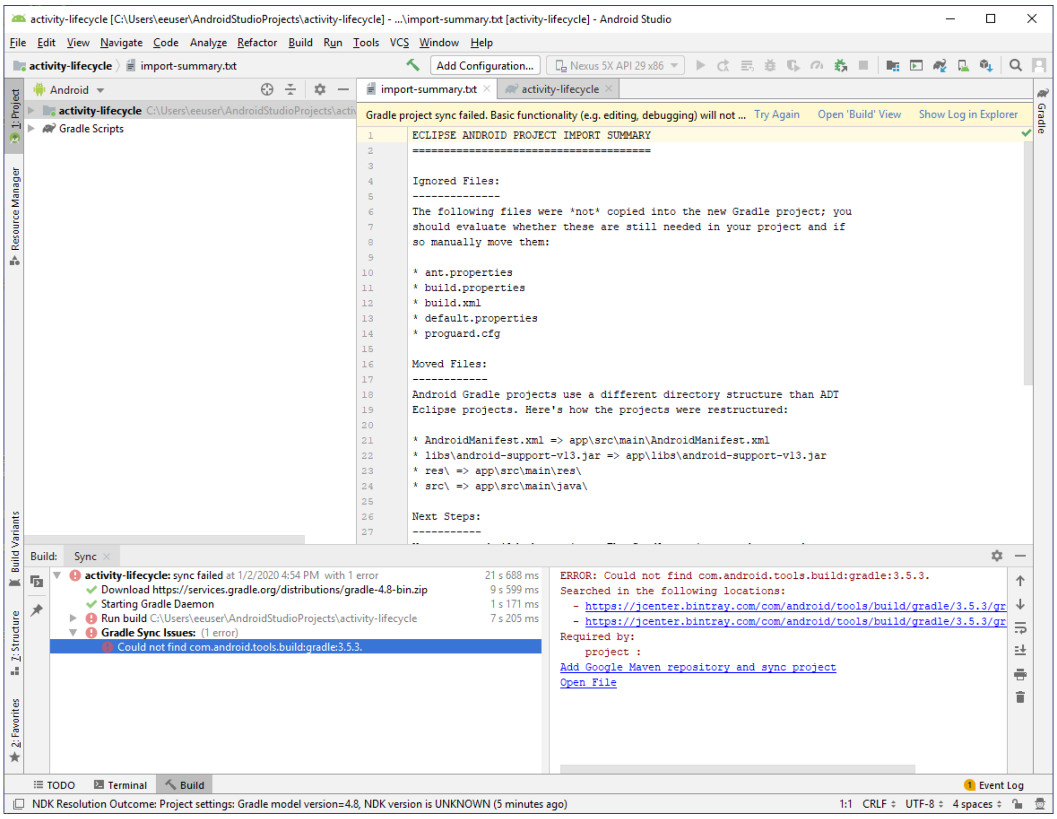


Import this demo project to your ADT by the wizard in **Import project (Gradle, Eclipse ADT, etc.)**


Android Studio
Version 3.0.1
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

The following figures show the required changes for making the ActivityLifecycle App to run.

The imported Activity Lifecycle app got the error message "Gradle project sync failed" similar to the following figure.



activity-lifecycle [C:\Users\eeuser\AndroidStudioProjects\activity-lifecycle] - ...\import-summary.txt [activity-lifecycle] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
activity-lifecycle 〉 import-summary.txt
Add Configuration...
Nexus 5X API 29 x86
Android
activity-lifecycle C:\Users\eeuser\AndroidStudioProjects\acti
Gradle Scripts
import-summary.txt
activity-lifecycle
1: Project
Resource Manager
Build Variants
7: Structure
2: Favorites
Gradle
TODO
Terminal
Build
Event Log

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not ... Try Again Open 'Build' View Show Log in Explorer

```
ECLIPSE ANDROID PROJECT IMPORT SUMMARY
======================================

Ignored Files:
--------------
The following files were *not* copied into the new Gradle project; you
should evaluate whether these are still needed in your project and if
so manually move them:

* ant.properties
* build.properties
* build.xml
* default.properties
* proguard.cfg

Moved Files:
------------
Android Gradle projects use a different directory structure than ADT
Eclipse projects. Here's how the projects were restructured:

* AndroidManifest.xml => app\src\main\AndroidManifest.xml
* libs\android-support-v13.jar => app\libs\android-support-v13.jar
* res\ => app\src\main\res\
* src\ => app\src\main\java\

Next Steps:
-----------
```

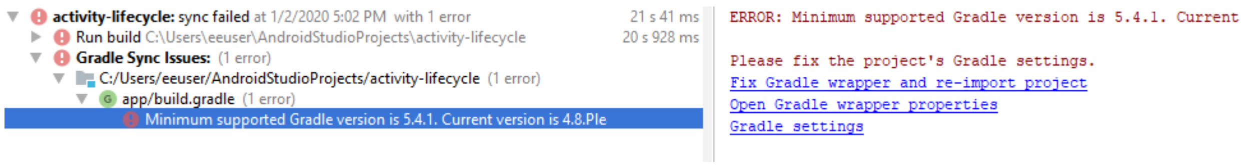
Build: Sync

```
activity-lifecycle: sync failed at 1/2/2020 4:54 PM  with 1 error              21 s 688 ms
  Download https://services.gradle.org/distributions/gradle-4.8-bin.zip        9 s 599 ms
  Starting Gradle Daemon                                                       1 s 171 ms
  Run build C:\Users\eeuser\AndroidStudioProjects\activity-lifecycle           7 s 205 ms
  Gradle Sync Issues: (1 error)
    Could not find com.android.tools.build:gradle:3.5.3.
```

```
ERROR: Could not find com.android.tools.build:gradle:3.5.3.
Searched in the following locations:
  - https://jcenter.bintray.com/com/android/tools/build/gradle/3.5.3/gr
  - https://jcenter.bintray.com/com/android/tools/build/gradle/3.5.3/gr
Required by:
    project :
Add Google Maven repository and sync project
Open File
```

NDK Resolution Outcome: Project settings: Gradle model version=4.8, NDK version is UNKNOWN (5 minutes ago)   1:1 CRLF UTF-8 4 spaces

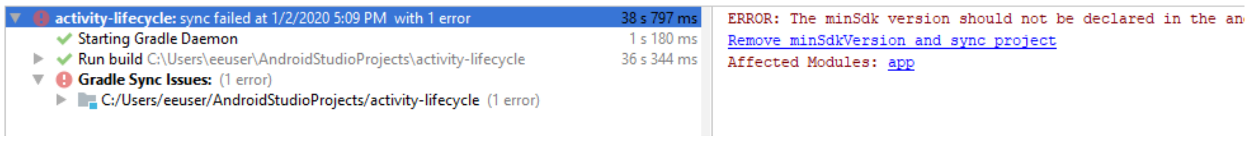
Click on the "Add Google Maven repository and sync project". After that, click on the "Do Refractor" located at the left bottom corner. Another error will be shown in the following figure,



```
activity-lifecycle: sync failed at 1/2/2020 5:02 PM  with 1 error           21 s 41 ms
  Run build C:\Users\eeuser\AndroidStudioProjects\activity-lifecycle        20 s 928 ms
  Gradle Sync Issues: (1 error)
    C:/Users/eeuser/AndroidStudioProjects/activity-lifecycle (1 error)
      app/build.gradle (1 error)
        Minimum supported Gradle version is 5.4.1. Current version is 4.8.Ple
```

```
ERROR: Minimum supported Gradle version is 5.4.1. Current

Please fix the project's Gradle settings.
Fix Gradle wrapper and re-import project
Open Gradle wrapper properties
Gradle settings
```

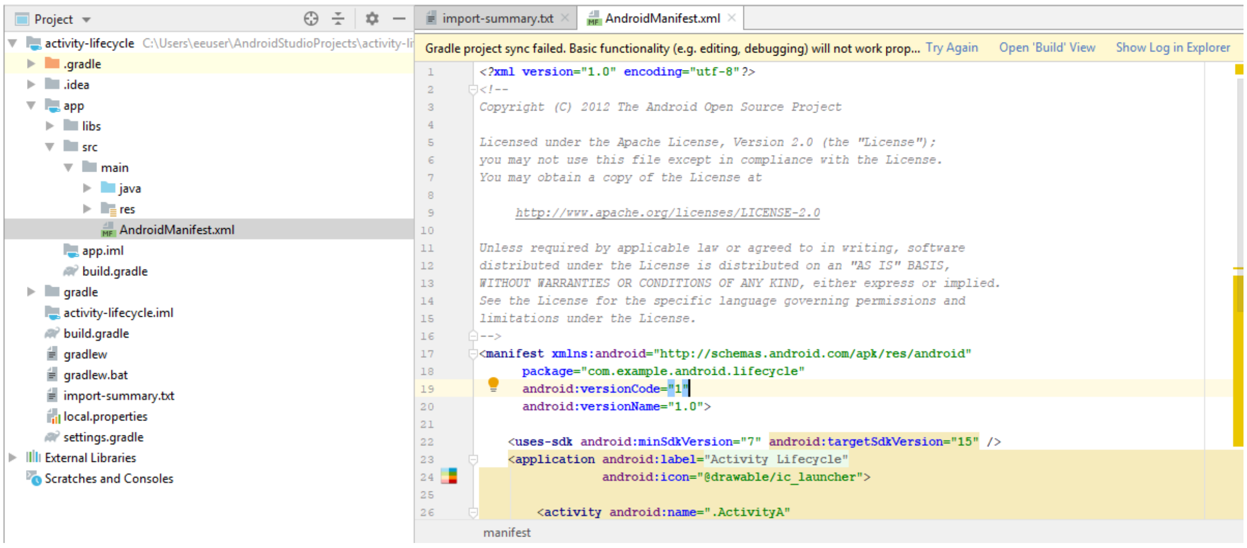
Click on the "Fix Gradle wrapper and re-import project", the following figure will result,



```
activity-lifecycle: sync failed at 1/2/2020 5:09 PM  with 1 error           38 s 797 ms
  Starting Gradle Daemon                                                    1 s 180 ms
  Run build C:\Users\eeuser\AndroidStudioProjects\activity-lifecycle        36 s 344 ms
  Gradle Sync Issues: (1 error)
    C:/Users/eeuser/AndroidStudioProjects/activity-lifecycle (1 error)
```

```
ERROR: The minSdk version should not be declared in the an
Remove minSdkVersion and sync project
Affected Modules: app
```

Select "Project View", open the "AndroidManifest.xml" file and remove the "android:minSdkVersion="7"" as shown in the following figure,



Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work prop...

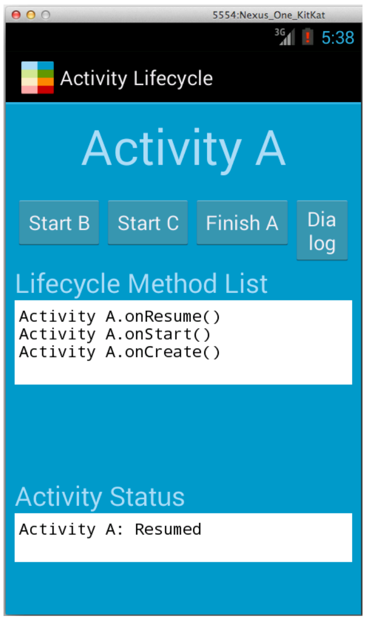
```
<?xml version="1.0" encoding="utf-8"?>
<!--
Copyright (C) 2012 The Android Open Source Project

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.example.android.lifecycle"
     android:versionCode="1"
     android:versionName="1.0">

    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="15" />
    <application android:label="Activity Lifecycle"
                 android:icon="@drawable/ic_launcher">

        <activity android:name=".ActivityA"
```

Then open the "build.gradle" file,

Change compileSdkVersion 14 --> 29

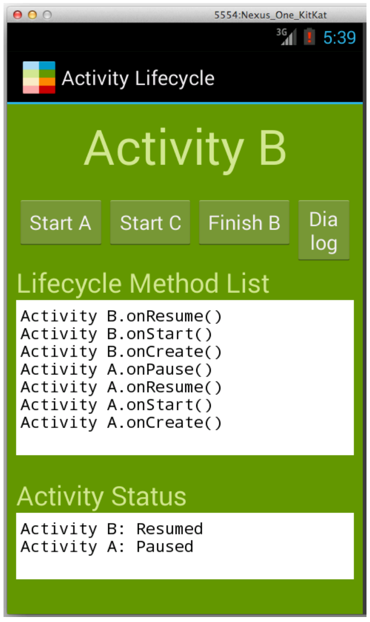
targetSdkVersion 15 --> 29

and click "try again". Then you can try to run the demo activity lifecycle app for experiencing the callback method sequences in the transition from one activity to another activity as shown below:



```
Activity A.onResume()
Activity A.onStart()
Activity A.onCreate()
```

```
Activity A: Resumed
```



```
Activity B.onResume()
Activity B.onStart()
Activity B.onCreate()
Activity A.onPause()
Activity A.onResume()
Activity A.onStart()
Activity A.onCreate()
```

```
Activity B: Resumed
Activity A: Paused
```

Check Point 2:

Show your "Activity Lifecycle" app.
Instructor Verification (separate page)

IV. Date and Time Pickers

To provide a widget for selecting a date, use the `DatePicker` widget, which allows the user to select the month, day, and year, in a familiar interface. In this section, students will create a `DatePickerDialog`, which presents the date picker in a floating dialog box at the press of a button. When the date is set by the user, a `TextView` will update with the new date.

1. Start a new project with application name of "**Date and Time Pickers**" with Activity Name of "`MainActivity`" and Layout Name of "`activity_main`".
2. Open the `res/layout/activity_main.xml` file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="5dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <DatePicker
        android:id="@+id/dpDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:datePickerMode="spinner"
        android:calendarViewShown="false"
        />

    <Button
        android:id="@+id/btnGet"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get date"
        android:onClick="getDate"/>

    <TimePicker
        android:id="@+id/tpTime"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:timePickerMode="spinner"
        />

</LinearLayout>
```

This creates a basic `LinearLayout` with a `DatePicker` and a `TimePicker` for letting user to select the date and time. A Button is used to show the selected date as Toasts message.

Basically, the DatePicker provides a widget for selecting the date consisting of day, month and year in your custom user interface. When the datePickerMode attribute is set to spinner, the date can be selected using year, month, and day spinners or a CalendarView. The set of spinners and the calendar view are automatically synchronized. The client can customize whether only the spinners, or only the calendar view, or both to be displayed.

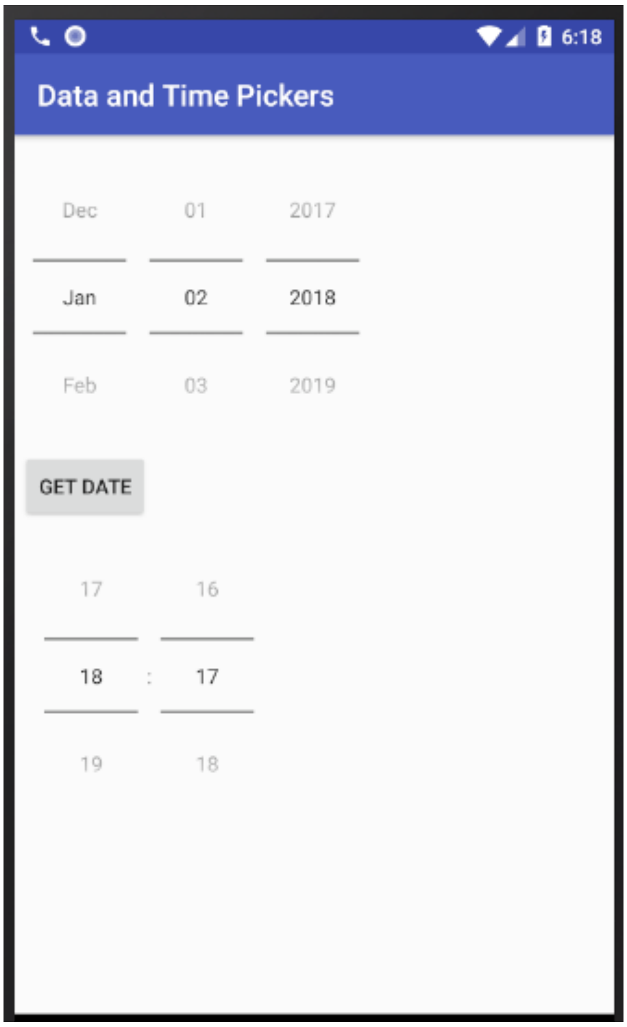
When the datePickerMode attribute is set to calendar, the month and day can be selected using a calendar-style view while the year can be selected separately using a list.

On the other hand, Android TimePicker provides a widget for selecting the time of day, in either 24-hour or AM/PM mode. You cannot select time by seconds. The timePickerMode attribute defines the look of the widget. It could be either clock or spinner. In order to get the time selected by the user on the screen, you will use `getCurrentHour()` and `getCurrentMinute()` method of the `TimePicker` class.

3. Modify the `MainActivity.java` with following code to handle the `DataPicker` and `TimePicker`:

```
public class MainActivity extends AppCompatActivity {

    DatePicker dpDate;
    TimePicker tpTime;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dpDate = (DatePicker)findViewById(R.id.dpDate);
        // init
        // dpDate.init(2002, 10, 27, null);


        tpTime = (TimePicker)findViewById(R.id.tpTime);

        // set the time picker mode to 24 hour view
        tpTime.setIs24HourView(true);

        // set a time changed listener to time picker
        tpTime.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {
            @Override
            public void onTimeChanged(TimePicker timePicker, int i, int i1) {
                String time = "Current time: " + timePicker.getCurrentHour() + " :
" + timePicker.getCurrentMinute();
                Toast.makeText(getApplicationContext(), time,
Toast.LENGTH_SHORT).show();
            }
        });

    }

    public void getDate(View v) {
        StringBuilder builder=new StringBuilder();
        builder.append("Current Date: ");
        builder.append((dpDate.getMonth() + 1)+"/");//month is 0 based
        builder.append(dpDate.getDayOfMonth()+"/");
        builder.append(dpDate.getYear());
        Toast.makeText(this, builder.toString(), Toast.LENGTH_SHORT).show();
    }
}
```

4. Run the application and then click on the button to show the selected date and change the spinner of the time picker to show the selected time.

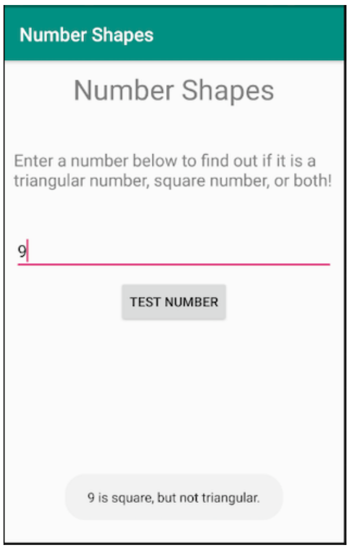


Check Point 3:

Show your “Date and Time Pickers” app.
Instructor Verification (separate page)

IV. Triangular and Square Testing App

In this section, students are required to build a sample app for testing a number is a triangular number, square number, or both with user interface as shown below:


Number Shapes
Number Shapes
Enter a number below to find out if it is a triangular number, square number, or both!
9
TEST NUMBER
9 is square, but not triangular.

The results of the number testing are shown as "Toast" message. To implement this app, a Java Class of testing triangular number and square number is provided as below:

```
class Number {
    int number;
    public boolean isSquare() {
        double squareRoot = Math.sqrt(number);
        if (squareRoot == Math.floor(squareRoot)) {
            return true;
        } else {
            return false;
        }
    }
    public boolean isTriangular() {
        int x = 1;
        int triangularNumber = 1;

        while (triangularNumber < number) {
            x++;
            triangularNumber = triangularNumber + x;
        }

        if (triangularNumber == number) {
            return true;
        } else {
            return false;
        }
    }
}
```

Students can use this class for your app implementation as well as understanding what is triangular and square numbers. Create a project with project name of "Number Shapes" and implement your app by modifying the MainActivity.java and activity_main.xml.

Check Point 4:

Show your "Number Shapes" app.
Instructor Verification (separate page)