Chapter 7.

Common Techniques for Android Apps Development

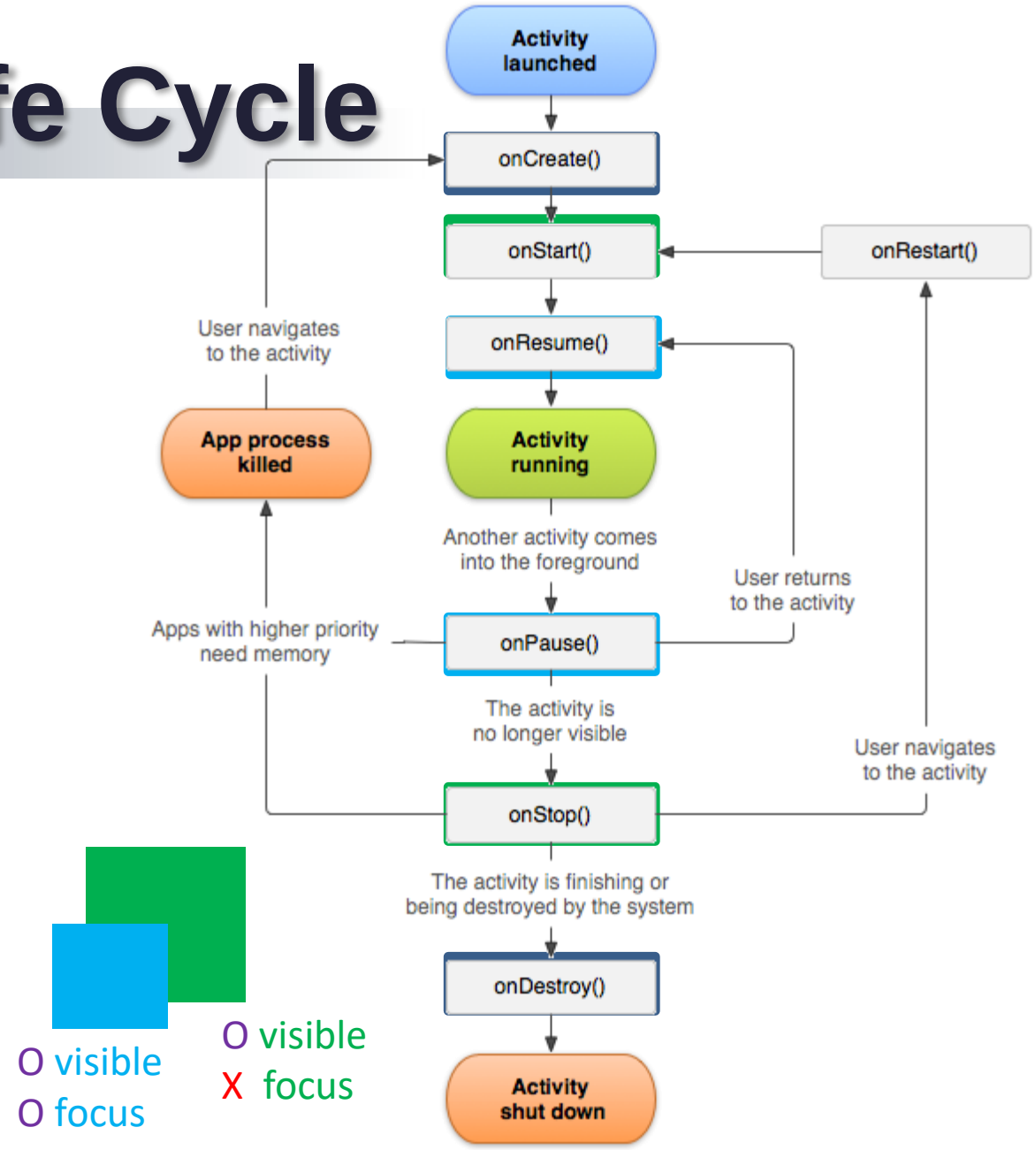
2023-2024
COMP7506 Smart Phone Apps Development
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

Agenda

- Android Activity Life Cycle
- Intent & Filter
  - Implicit Intent Examples
  - Explicit Intent Examples
  - Passing Extra Data
- Concept of Service
  - “startService()” Approach
  - “bindService()” Approach
- Data Storage Issues
- Simple Graphics
  - Touch Screen & Dragging
- Audio Playing

Activity Life Cycle

- onCreate()
  - Enter memory
- onStart()
  - Become visible
- onResume()
  - Get focus / foreground
- onPause()
  - Lose focus / foreground
- onStop()
  - Become invisible
- onDestroy()
  - Leave memory



Note: An activity can be visible but not in focus if the phone screen can display multiple activities at the same time.

Activity Coordination (Example)

- Activity A starts Activity B:
  - 1. A's onPause() method executes.
  - 2. B's onCreate(), onStart(), and onResume() methods execute in sequence. (B now has user focus.)
  - 3. If A is no longer visible on screen, its onStop() method executes.
- So, e.g., if A needs to write to a DB before B starts, use onPause() instead of onStop().

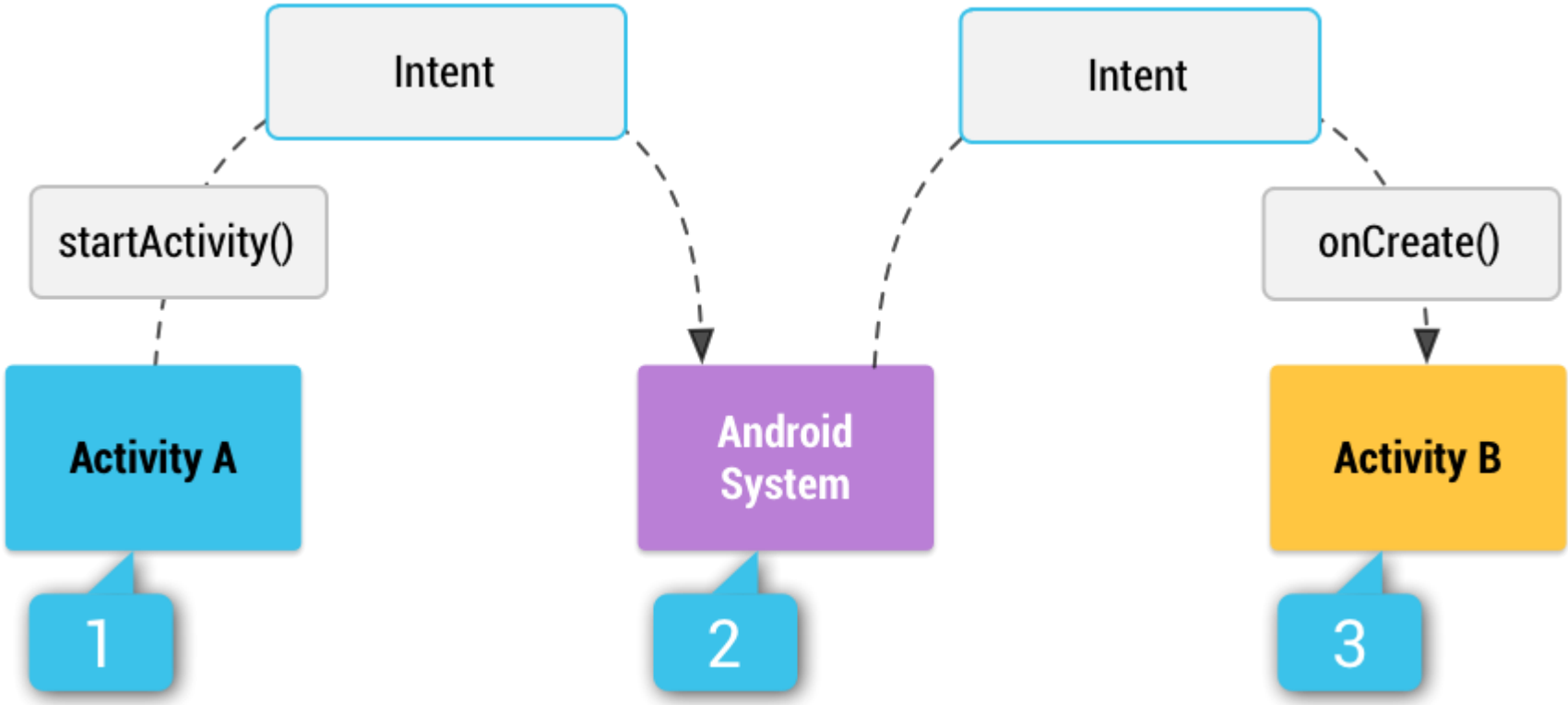
Intent & Filter

Android intents can be imagined as object passing between activities, services or receivers.

Explicit vs. Implicit Intent

- Explicit
  - Explicit intents are used in the application itself wherein one activity can switch to another activity.
  - E.g., Intent intent = new Intent(this, FooActivity.class); startActivity(intent);
- Implicit
  - Implicit intents do not directly specify the Android components which should be called , it only specifies action to be performed.
  - E.g., Intent intent = new Intent(Intent.ACTION_SEND); intent.putExtra(Intent.EXTRA_EMAIL, recipientArray); startActivity(intent);

Implicit Intent Handling



Intent Filter (for Implicit Intents only)

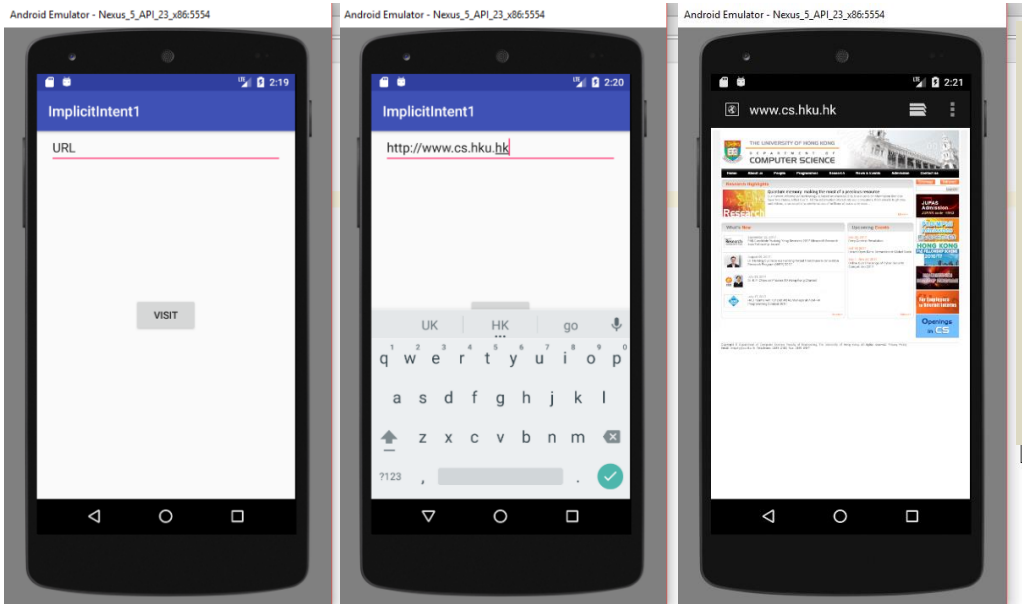
- Android determines the best activity or service (or set of receivers) on its own.
- It does so by comparing the contents of the Intent object to intent filters, structures associated with components that can potentially receive intents.
- Filters advertise the capabilities of a component and delimit the intents it can handle. They open the component to the possibility of receiving implicit intents of the advertised type.
- If a component does not have any intent filters, it can receive only explicit intents. A component with filters can receive both explicit and implicit intents.

Implicit Intent Example

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
startActivity(intent);
```

Intent.ACTION_VIEW: To display some information

Uri.parse(url): Uniform Resource Identifier (Uniform Resource Locator)



Intent filter in Android browser apps:

```
<activity android:name="hkucs.sample.Activity2">
    <intent-filter>
        <action android:name = "android.intent.action.VIEW" />
        <action android:name = "Intent.ACTION_VIEW" />
        ……
    </intent-filter>
</activity>
```

Explicit Intent Example

- In Android, we usually use explicit intent to do page / activity switching.
- Key steps:
  - Create a new layout for the new activity
  - Define the new activity in manifest file
  - In the calling activity (old page), jump to the new activity (new page) by creating a new Intent instance and starting it.
  - In the called activity (new page), return to the calling activity (old page) by creating another Intent instance.

Explicit Intent Example

Definition of activities in AndroidManifest.xml:

```
<activity android:name=".MainActivity1" >
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
<activity android:name=".MainActivity2">
</activity>
```

An activity having this intent filter will become the first activity (i.e., being loaded upon the app is started).

New activity

Steps in First & Second Activities

- **First Activity:**
  - Creates an "Intent" to start the second activity. The intent needs two parameters: a context and the name of the second activity (e.g., MainActivity2.class).

    ```
    Intent myIntent = new Intent(view.getContext(), MainActivity2.class);
    ```
  - Start the activity using startActivity().

    ```
    startActivity(myIntent);
    ```
- **Second Activity:**
  - Creates an "Intent" to start the first activity. The intent needs two parameters: a context and the name of the first activity (e.g., MainActivity1.class).

    ```
    Intent myIntent = new Intent(view.getContext(), MainActivity1.class);
    ```
  - Start the activity using startActivity().

    ```
    startActivity(myIntent);
    ```

Passing Extra Data

Sender Activity

- To pass more data from the first activity to the second activity, you may use the following:

```
Intent myIntent =
        new Intent(view.getContext(), MainActivity2.class);
myIntent.putExtra("ID", 7506);
startActivity(myIntent);
```

Receiver Activity

- In the second activity, you can obtain the value of ID as follows:
  - Approach 1:

```
Intent myIntent = getIntent();
int id = myIntent.getIntExtra("ID", 0);   // id = 7506
```

  - Approach 2:

```
Bundle extras = getIntent().getExtras();
int id = extras.getInt("ID");   // id = 7506
```

- Can use getStringExtra("ID") / getString("ID") if ID is a string.
- Can use getLongExtra("ID", 0) / getLong("ID") if ID is a long integer.

Passing Arrays and Objects

You can pass arrays and objects between intents.

```
public class MainActivity1 extends AppCompatActivity {
        private int[] myIntArray;
        private String[] myStrArray;
        private Calendar myCal;
        …
        public void onClick(View view) {
                Intent intent = new Intent(view.getContext(), MainActivity2.class);
                intent.putExtra("IntArray", myIntArray);
                intent.putExtra("StrArray", myStrArray);
                intent.putExtra("Calendar", myCal);
                …
```

Passing Arrays and Objects

There are two approaches for receiving the arrays and objects.

Approach 1:

```
public class MainActivity2 extends AppCompatActivity {
        private int[] myIntArray;
        private String[] myStrArray;
        private Calendar myCal;
        …
        public void onClick(View view) {
                Intent intent = getIntent();
                myIntArray = intent.getIntArrayExtra("IntArray");
                myStrArray = intent.getStringArrayExtra("StrArray");
                myCal = (Calendar) intent.getSerializableExtra("Calendar");
                …
```

Serialization is the conversion of an object to a series of bytes, so that the object can be easily saved to persistent storage or streamed across a communication link or from one virtual machine to another. The byte stream can then be desterilized - converted into a replica of the original object.

Passing Arrays and Objects

Approach 2:

```
public class MainActivity2 extends AppCompatActivity {
        private int[] myIntArray;
        private String[] myStrArray;
        private Calendar myCal;
        …
        public void onClick(View view) {
                Bundle extras = getIntent().getExtras();
                myIntArray = extras.getIntArray(“IntArray”);
                myStrArray = extras.getStringArray(“StrArray”);
                myCal = (Calendar) extras.getSerializable(“Calendar”);
                …
```

Check Intent Recipient

- Your app will crash if the intent has no recipient.
- You better check for exception.

```
try {
        startActivity(intent);
}
catch(android.content.ActivityNotFoundException e) {
        …
}
```

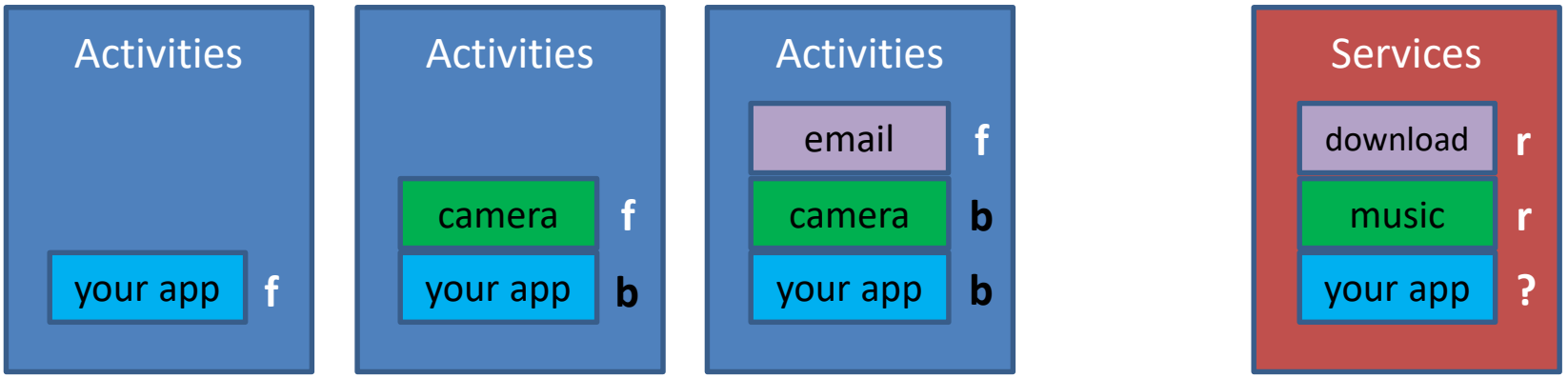
Recall: try-catch block is used for risky operations in Java.

Concept of Service

We seldom emphasize the term “service” (or background task) in PC or server environment. However, we must emphasize the difference between foreground tasks (activities) and background tasks (services) in mobile environment because mobile users care the UI performance a lot.

Service

- Long running operations in the background
- Usage similar to activity (component + intent), but
  - No user interface (invisible)
    - Activities are visible with UI
  - Can have multiple services running
    - e.g., downloading files, playing music
    - Only one activity can be *running*



Forms

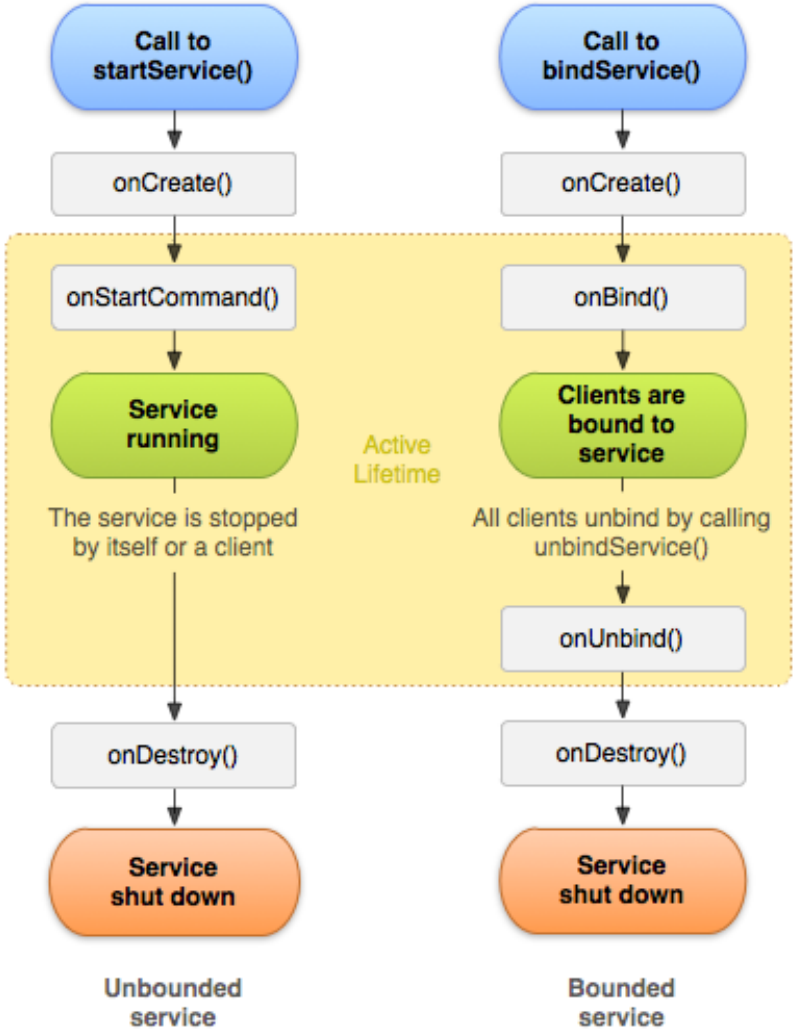
- Started
  - startService()
  - Can run indefinitely even if caller app dies
  - No return result
  - Simple, single task
- Bound
  - bindService()
  - Can bind to multiple components (polygamy)
  - Terminates if all callers die
  - Send request, get result, IPC
- Can be both
  - e.g., started, and get bound
  - onStartCommand() / onBind() allows start / bind

Analogy

- The instructor asks the TA to go out and buy a cup of coffee.
  - startService():
    - Once the TA goes, the instructor has no control on him.
    - The TA can play Pokémon Go, go to washroom and take a cigarette before going to buy a cup of coffee.
    - The instructor cannot change his mind.
  - bindService():
    - Before the TA goes, he needs to install an app on his mobile phone.
    - The instructor can keep track of the TA's current location.
    - The instructor can even instruct the TA to buy a cup of milk instead of a cup of coffee.

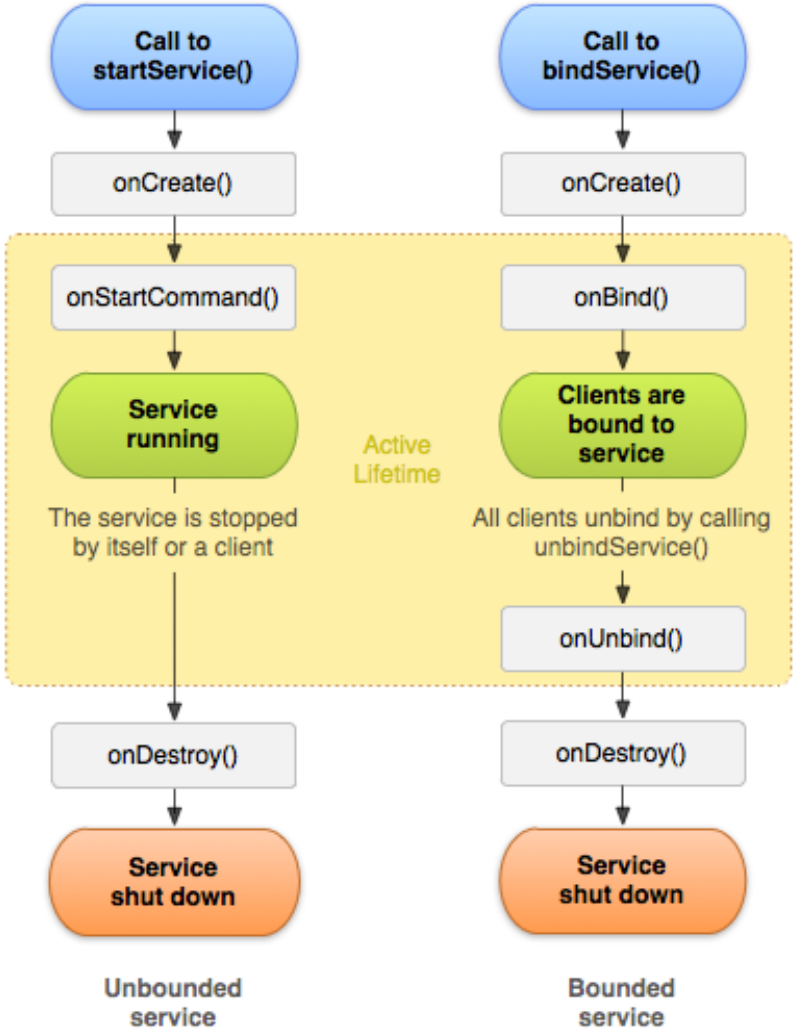
Service Life Cycle

- 2 approaches to start a service:
  - startService()
  - bindService()
- onCreate()
  - One time setup
  - Analogous to activity
- onDestroy()
  - Clean up resources



Service Life Cycle

- onStartCommand()
  - In response to startService()
  - Service runs indefinitely
  - So requires explicit stop
    - stopService()
    - stopSelf() (e.g., used with a timer)
- onBind()
  - In response to bindService()



Force Stop

- Happens only if system resource is low
- Depending on priorities
  - Bound to foreground activity
  - Declared to be in foreground
  - Short running
  - Long running

- Restart later when resources become available
- Need to handle this carefully
  - Processes & threads

Service Definition in Manifest

- Similar to activity

```
<application
        <activity android:name=".ExampleActivity" >
                <intent-filter >
                …
                </intent-filter>
        </activity>
        <service android:name=".ExampleService" >
        …
        </service>
</application>
```

“startService()” Approach

For reference only, no need to memorize!

Approach 1: startService() Syntax

- Define a sub-class of Service and override the following 3 methods:
  - public IBinder onBind(Intent intent) { return null; }
    - Not used in this approach and so just return null
  - public int onStartCommand(Intent intent, int flags, int startId) { }
    - The 3 parameters are from the system and may not be useful
    - Called when the service is started
  - public void onDestroy() { }
    - Called when the service is stopped
- Define the service in Android Manifest:
  - <service android:name=".XXX" /> where XXX is the sub-class name

Approach 1: startService() Syntax

- To start the service from the main program, call the following:
  - startService(new Intent(getBaseContext(), XXX.class)); where XXX is the name of the Service sub-class
    - onStartCommand() in Service sub-class will be called
- To stop the service from the main program, call the following:
  - stopService(new Intent(getBaseContext(), XXX.class)); where XXX is the name of the Service sub-class
    - onDestroy() in Service sub-class will be called

Approach 1: startService() Example (MyService.java)

```
public class MyService extends Service {
        @Nullable
        @Override
        public IBinder onBind(Intent intent) {
                return null;
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
                // Let it continue running until it is stopped.
                Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
                return START_STICKY;
        }

        @Override
        public void onDestroy() {
                super.onDestroy();
                Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
        }
}
```

- A toast is a view containing a quick little message for the user. The toast class helps you create and show those.
- LENGTH_LONG: Show the view or text notification for a long period of time (~ 5 sec).
- LENGTH_SHORT: Show the view or text notification for a short period of time (~ 3 sec).

Approach 1: startService() Return Value of onStartCommand()

```
public int onStartCommand(Intent intent, int flags, int startId) {
        // do stuff
        return START_STICKY;
}
```

- Return value determines behavior after the service killed:
  - START_NOT_STICKY
    - Do not recreate after kill
    - Caller can restart unfinished jobs
  - START_STICKY
    - Recreate, but do not redeliver last intent
    - Continuous work but stateless, e.g., media players
  - START_REDELIVER_INTENT
    - Recreate, and redeliver last intent
    - Actively performing a job, e.g., file download

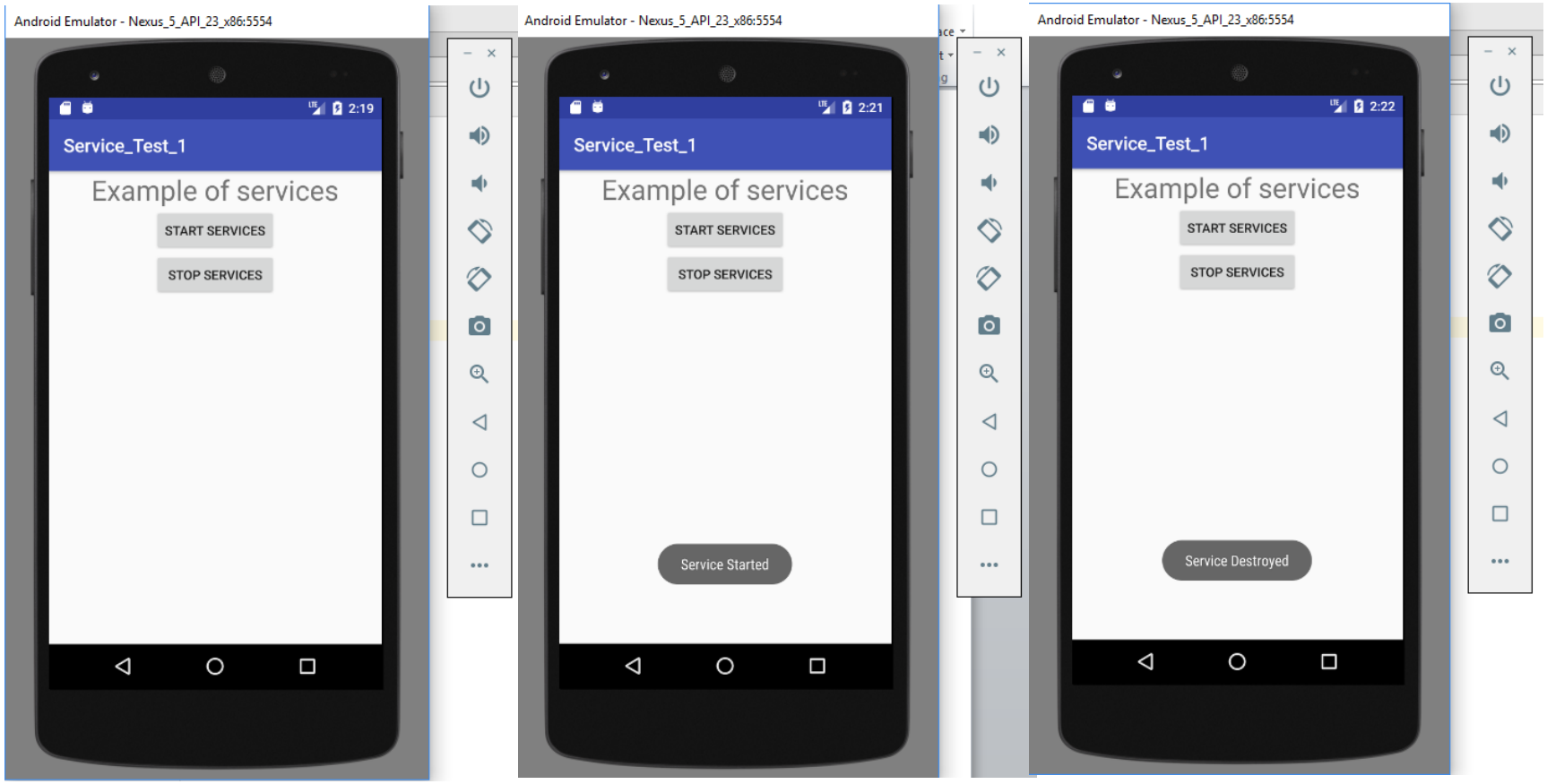
Approach 1: startService() Example (MainActivity.java)

```
public class MainActivity extends AppCompatActivity {
        String msg = "Android : ";

        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
        }
        public void startService(View view) {
                startService(new Intent(getBaseContext(), MyService.class));
        }

        // Method to stop the service
        public void stopService(View view) {
                stopService(new Intent(getBaseContext(), MyService.class));
        }
}
```

Layout file:

```
<Button
...
android:text="Start Services"
android:onClick="startService"
... />

<Button
android:text="Stop Services"
android:onClick="stopService"
... />
```

Approach 1: startService() Sample Output



Please try "ServiceStart" for the full example.

Wake Lock

- A service will be suspended if the Android phone goes asleep. In order to keep the phone's CPU running, you can:
  - Step 1:
    - Add the following permission statement into Manifest file:

```
<users-permission android:name="android.permission.WAKE_LOCK" />
```

  - Step 2:
    - Add the following to the onCreate() method of service class:

```
PowerManager powerManager =
        (PowerManager) getSystemService(Context.POWER_SERVICE);
PowerManager.WakeLock wakeLock = powerManager.newWakeLock
        (PowerManager.PARTIAL_WAKE_LOCK, "MyWakeLock");
wakeLock.acquire();
```

    - Add the following to the onDestroy() method of service class:

```
wakeLock.release();
```

“bindService()” Approach

For reference only, no need to memorize!

Approach 2: bindService() Syntax

- Define a sub-class of Service MyService and do the following:
  - Override the following methods:
    - public IBinder onBind(Intent intent) { … }
    - public boolean onUnbind(Intent intent) { … }
    - public void onCreate() { … }
    - public void onDestroy() { … }
  - Define a label with the type IBinder and it will be used to point to a Binder object:
    - private final IBinder binder = new ServiceBinder();
  - Define an inner sub-class of Binder:

```
public class ServiceBinder extends Binder {
        MyService getService() {
                return MyService.this;
        }
}
```

Approach 2: bindService() Syntax

- In the main program:
  - Define a label with the type as the service sub-class:
    - private MyService myService; where MyService is the name of the service sub-class
  - Define the ServiceConnection object:
    - private ServiceConnection serviceCon = new ServiceConnection() {

```
@Override
public void onServiceConnected(ComponentName className, IBinder binder) {
        myService = ((MyService.ServiceBinder) binder).getService();
}
@Override
public void onServiceDisconnected(ComponentName className) {
        myService = null;
}
};
```

  - Define an Intent object for the service class and call bindService() to bind it:
    - Intent intent = new Intent(this, MyService.class);
    - bindService(intent, serviceCon, Context.BIND_AUTO_CREATE);

Approach 2: bindService() Example (MusicService.java)

```
public class MusicService extends Service {
        private final IBinder binder = new ServiceBinder();
        public String play() {
                return getString(R.string.msg_musicPlay);
        }
        public String stop() {
                return getString(R.string.msg_musicStop);
        }
        @Override
        public void onCreate() {
                super.onCreate();
        }
        @Override
        public IBinder onBind(Intent intent) {
                Toast.makeText(this, "onBind",
                        Toast.LENGTH_SHORT).show();
                return binder;
        }
        public class ServiceBinder extends Binder {
                MusicService getService() {
                        return MusicService.this;
                }
        }
        @Override
        public boolean onUnbind(Intent intent) {
                Toast.makeText(this, "onUnbind",
                        Toast.LENGTH_SHORT).show();
                return false;
        }

        @Override
        public void onDestroy() {
                super.onDestroy();
        }
        }
```

Approach 2: bindService() Example (MainActivity.java)

```
public class MainActivity extends AppCompatActivity {
        private TextView tvMessage;
        private Button btPlay, btStop;
        private boolean isBound;
        private MusicService musicService;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main_activity);
                findViews();
        }
        private void findViews() {
                tvMessage = (TextView) findViewById(R.id.tvMessage);
                btPlay = (Button) findViewById(R.id.btPlay);
                btStop = (Button) findViewById(R.id.btStop);
                btPlay.setVisibility(View.INVISIBLE);
                btStop.setVisibility(View.INVISIBLE);
        }
        public void onConnectClick(View view) {
                doBindService();
        }
```

Layout file:

```
<Button
...
android:text="Connect Service"
android:onClick="onConnectClick" />
<Button
...
android:text="Disconnect Service"
android:onClick="onDisconnectClick" />
<Button
...
android:text="Play Music"
android:onClick="onPlayClick" />
<Button
…
android:text="Stop Music"
android:onClick="onStopClick" />
```

Approach 2: bindService() Example (MainActivity.java)

```
public void onDisconnectClick(View view) {
        doUnbindService();
}
public void onPlayClick(View view) {
        String message = musicService.play();
        tvMessage.setText(message);
}
public void onStopClick(View view) {
        String message = musicService.stop();
        tvMessage.setText(message);
}
void doBindService() {
        if (!isBound) {
                Intent intent = new Intent(this, MusicService.class);
                bindService(intent, serviceCon, Context.BIND_AUTO_CREATE);
                isBound = true;
        }
}
@Override
public void onDestroy() {
        super.onDestroy();
        doUnbindService();
}
```

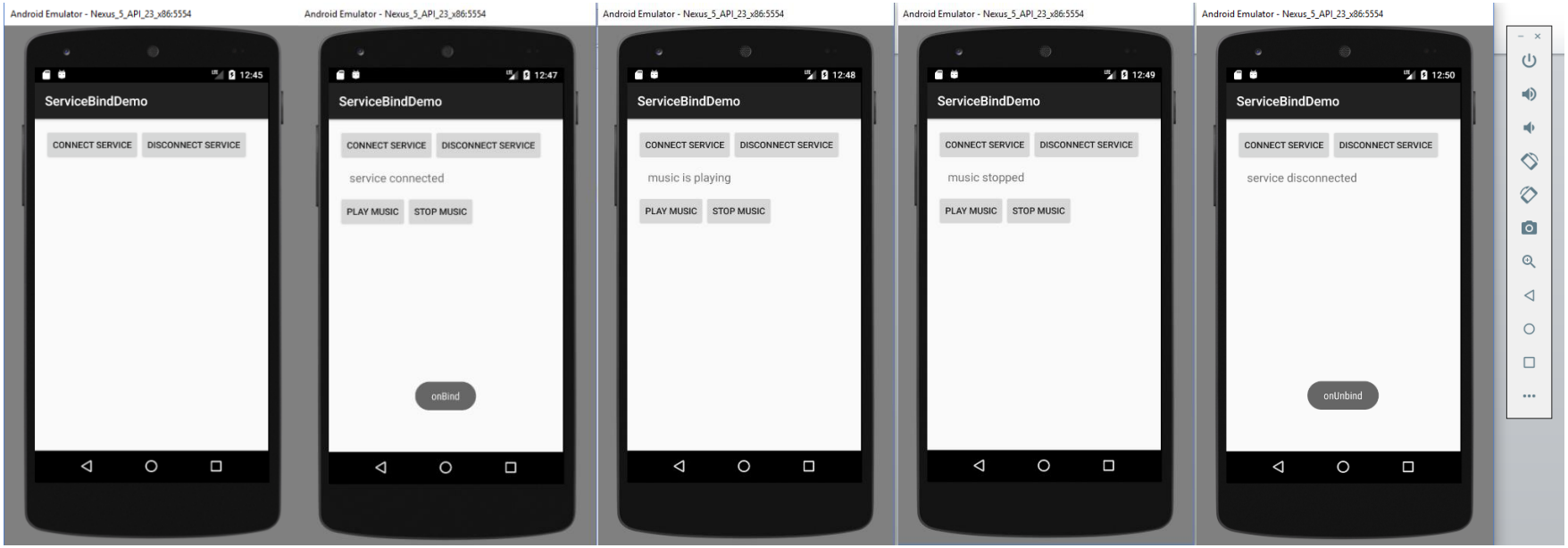
Layout file:

```
<Button
...
android:text="Disconnect Service"
android:onClick="onDisconnectClick" />
<Button
...
android:text="Play Music"
android:onClick="onPlayClick" />
<Button
…
android:text="Stop Music"
android:onClick="onStopClick" />
```

Approach 2: bindService() Example (MainActivity.java)

```
void doUnbindService() {
        if (isBound) {
                unbindService(serviceCon);
                isBound = false;
                btPlay.setVisibility(View.INVISIBLE);
                btStop.setVisibility(View.INVISIBLE);
                tvMessage.setText(R.string.msg_serviceDisconnected);
        }
}
private ServiceConnection serviceCon = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className, IBinder binder) {
                musicService = ((MusicService.ServiceBinder) binder).getService();
                tvMessage.setText(R.string.msg_serviceConnected);
                btPlay.setVisibility(View.VISIBLE);
                btStop.setVisibility(View.VISIBLE);
        }
        @Override
        public void onServiceDisconnected(ComponentName className) {
                musicService = null;
                tvMessage.setText(R.string.msg_serviceLostConnection);
        }
}; }
```

Approach 2: bindService()
Sample Output



Please try "ServiceBind" for the full example.

Data Storage Issues

Data Storage

- Is it possible for an application to share data with another? For example, a company may publish multiple apps and suppose all of them require user profiles. Is it possible for the apps to share a single user profile?
- Is it possible for an activity to share data with another without putting extra data into an intent object?

Data Storage

- Possible approaches:
  - SharedPreferences
    - Private primitive data in key-value pairs
  - File I/O
    - Private data on device memory

SharedPreferences

- Persistent (not affected even you close the application or even switch off the Android phone) key-value pairs of primitive data types
  - e.g., “textEntry” (key) + “foobar” (value)
  - Primitive types: Boolean, Int, Long, String
- Read
  - getSharedPreferences(PREFS_NAME, …).get*()
  - *: primitive types
- Write
  - getSharedPreferences(PREFS_NAME, …).edit().put*()

Example

```
public class Calc extends AppCompatActivity {
        public static final String PREFS_NAME = "MyPrefsFile";
        @Override
        protected void onCreate(Bundle state){
                super.onCreate(state);

                // Restore preferences
                SharedPreferences settings = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
                boolean silent = settings.getBoolean("silentMode", false);
                setSilent(silent);
        }
        @Override
        protected void onStop(){
                super.onStop();
                // We need an Editor object to make preference changes.
                // All objects are from android.context.Context
                SharedPreferences settings = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
                SharedPreferences.Editor editor = settings.edit();
                editor.putBoolean("silentMode", mSilentMode);
                // Commit the edits!
                editor.commit();
        }
}
```

File I/O

- Only available to your application by default
  - Be removed when app is uninstalled
- Input
  - openFileInput()
  - read()
  - close()
- Output
  - openFileOutput()
  - write()
  - close()

File I/O (Syntax)

- `String FILENAME = "hello_file";`
- `String string = "hello world!";`

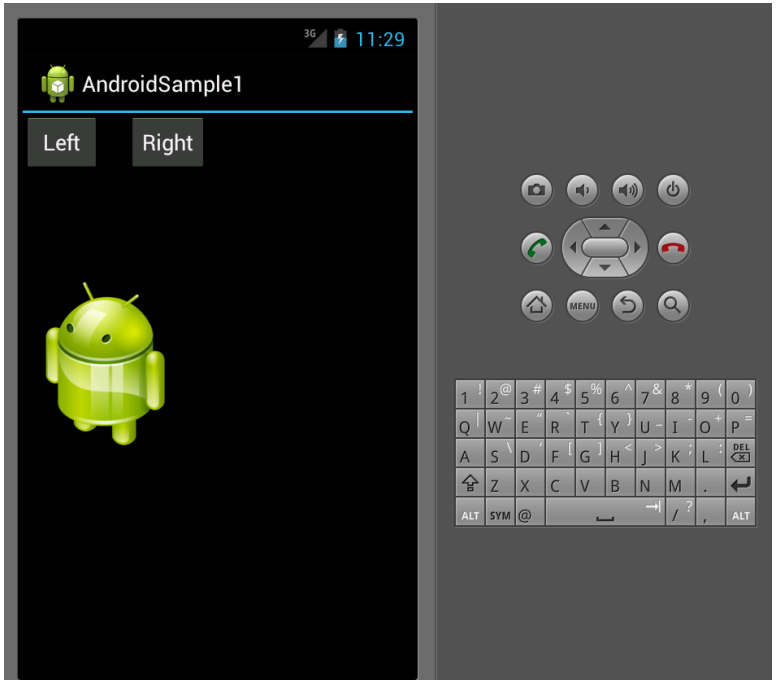
- `FileInputStream fin = openFileInput(FILENAME);`
- `int c = fin.read();`
- `fin.close();`

- `FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);`
- `fos.write(string.getBytes());`
- `fos.close();`

Simple Graphics

Moving an Image

- Our objective: Design an Android application such that an image can be moved by pressing “Left” or “Right” buttons.



Referring to Components

- Open and edit the main program
- Note: Recall “btnLeft”, “btnRight” and “image1” are identities of the 3 components we defined.

```
public class MainActivity extends AppCompatActivity {

Button button1, button2;
ImageView image1;

public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        image1 = (ImageView) findViewById(R.id.image1);
        button1 = (Button) findViewById(R.id.btnLeft);
        button2 = (Button) findViewById(R.id.btnRight);
```

Layout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/
        apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <Button
        android:id="@+id/btnLeft"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Left" />
    <Button
        android:id="@+id/btnRight"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="22dp"
        android:layout_toRightOf="@+id/btnLeft"
        android:text="Right" />
    <ImageView
        android:id="@+id/image1"
        android:layout_width="135dp"
        android:layout_height="135dp"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/btnLeft"
        android:layout_marginTop="89dp"
        android:src="@drawable/android3d" />
</RelativeLayout>
```

Moving Left

- Condition: Left margin of the image should be always greater than 0 (to ensure that the image is always on the screen).
- We use “LayoutParams” to get the parameters of the image on the RelativeLayout, update it and set it again.
- You have to include “android.widget.RelativeLayout” in order to use “LayoutParams”.

```
button1.setOnClickListener(new View.OnClickListener() {
        public void onClick(View arg0) {
                RelativeLayout.LayoutParams par
                = (RelativeLayout.LayoutParams) image1.getLayoutParams();
                if (par.leftMargin – 5 > 0) {
                        par.leftMargin -= 5;
                }
                image1.setLayoutParams(par);
        }
});
```

Moving Right

- Condition: Right margin of the image should be always smaller than the width of the screen (to ensure that the image is always on the screen).
- We need to make use of the “android.view.Display” library.
- You have to include “android.view.Display” in order to use “Display”.

```
button2.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0) {
        Display display = getWindowManager().getDefaultDisplay();
        Point size = new Point();
        display.getSize(size);
        int screen_width = size.x;
        int screen_height = size.y;
```

Point can be treated as a structure with 2 components.

Moving Right

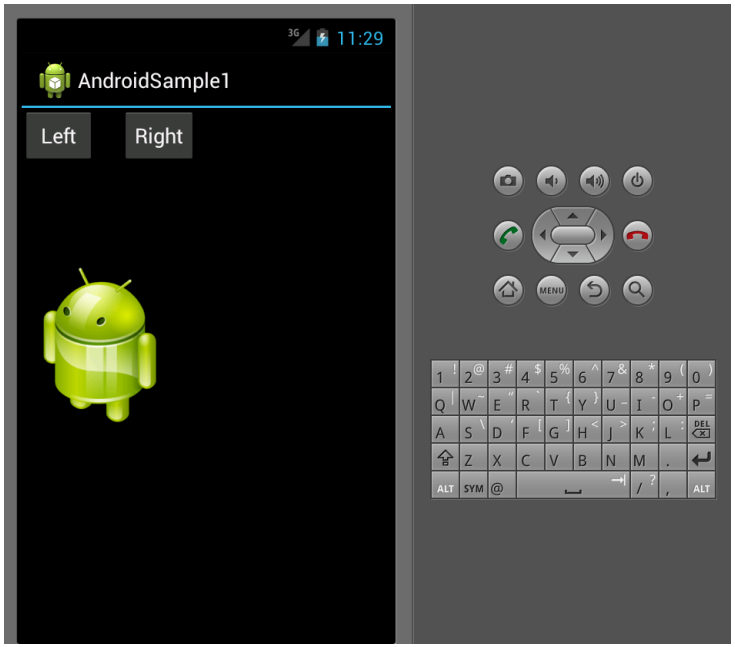
- We use “LayoutParams” to get the parameters of the image on the RelativeLayout, update it and set it again.

```
                RelativeLayout.LayoutParams par
                = (RelativeLayout.LayoutParams) image.getLayoutParams();
                if (par.leftMargin + 5 < screen_width - par.width) {
                        par.leftMargin += 5;
                }
                image.setLayoutParams(par);
        }
});
```

Touch Screen & Dragging

Dragging an Image

- Our objective: Design an Android application such that an image can be dragged to move using the touch screen.



Touch Screen

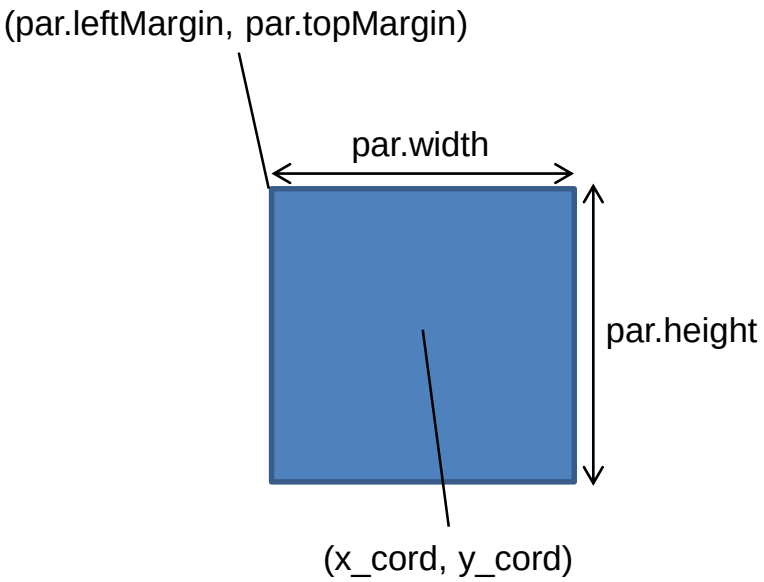
- We need to set the “OnTouchListener” of the image.
- You have to include “android.view.MotionEvent” in order to use touch screen.

```
image.setOnTouchListener(new View.OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        LayoutParams par = (LayoutParams) image.getLayoutParams();
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            ...
            break;
        case MotionEvent.ACTION_MOVE:
            ...
            break;
        default:
            ...
            break;
        }
        return true;
    }
});
```

Touch Screen

Setting the moving logic by referencing the location of the touch point (can be obtained using “event.getRawX()” and “event.getRawY()”).

```
case MotionEvent.ACTION_MOVE:
        int x_cord = (int)event.getRawX();
        int y_cord = (int)event.getRawY();
        par.leftMargin = x_cord - par.width / 2;
        par.topMargin = y_cord - par.height / 2;
        image.setLayoutParams(par);
        break;
```



Ensure that you are touching the center of the image.

Audio Playing

Playing Background Music

- Supports common audio formats such as *.mid and *.mp3
- To be placed into the folder “res/raw”
- “raw” is a default sub-folder under “res” in Eclipse. In Android Studio, you need to create it on your own.
- Needs to import the library “android.media.MediaPlayer”

Playing Background Music

- Create a variable of the type “MediaPlayer”:

```
MediaPlayer mediaPlayer;
```

- Create an instance and initialize it with your background music file:

```
mediaPlayer = MediaPlayer.create(this, R.raw.xxx);
```

- Start playing music:

```
if(!mediaPlayer.isPlaying())
        mediaPlayer.start();
```

Playing Background Music

- Pause playing music:

```
if(mediaPlayer.isPlaying())
        mediaPlayer.pause();
```

- Stop playing music:

```
if(mediaPlayer.isPlaying())
        mediaPlayer.stop();
```

Playing Sound Effect

- Supports common audio formats such as *.mid, *.mp3 and *.ogg but the sound files played with SoundPool should not exceed 1 MB.
- *.ogg files are played better by Android.
- To be placed into the folder “res/raw”
- Needs to import the library “android.media.SoundPool”

Playing Sound Effect

- Create a variable of the type “SoundPool”:

```
SoundPool soundPool;
```

- Create a variable of the type “HashMap<Integer, Integer>” for storing the sounds once they are loaded:

```
HashMap<Integer, Integer> soundPoolMap;
```

- Create an instance of SoundPool
  - Below Android 5.0:

```
soundPool = new SoundPool(4, AudioManager.STREAM_MUSIC, 100);
```

    Note: The first parameter states how many sounds can be played at once. If you try to play more than this number then it stops the oldest stream. The second parameter states the type of sound. The third parameter states the sound quality.

  - Android 5.0 or above:

```
soundPool = new SoundPool.Builder()
            .setMaxStreams(4)
            .build()
```

    Note: setMaxStreams() is for setting how many sounds can be played at once. If you try to play more than this number then it stops the oldest stream. You don’t need to state the type of sound and the sound quality.

Support of old and new versions

- If you want to support old versions of Android, you can check the API version.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        mSoundPool = new SoundPool.Builder()
                .setMaxStreams(4)
                .build();
} else {
        mSoundPool = new SoundPool(4, AudioManager.STREAM_MUSIC, 100);
}
```

Playing Sound Effect

- Create an instance of HashMap<Integer, Integer>:

```
soundPoolMap = new HashMap<Integer, Integer>();
```

- Put your sound effect file into the hash map:

```
soundPoolMap.put(soundID, soundPool.load(this, R.raw.xxx, 1));
```

  Note: The last parameter states the index of the sound effect you want to play.

Playing Sound Effect

- Create an AudioManager to handle the service that plays the sound effect:

```
AudioManager audioManager =
(AudioManager)getSystemService(Context.AUDIO_SERVICE);
```

- Play the sound effect:

```
soundPool.play(soundID, leftVolume, rightVolume, priority, no_loop,
normal_playback_rate);
```

Playing Sound Effect

where the parameters can be set as follows:

```
float curVolume =
audioManager.getStreamVolume(AudioManager.STREAM_MUSIC);
float maxVolume =
audioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
float leftVolume = curVolume/maxVolume;
float rightVolume = curVolume/maxVolume;
int priority = 1;
int no_loop = 0; // set to 1 if you want the sound effect to loop
float normal_playback_rate = 1f;
```

Chapter 7.

End

2023-2024
COMP7506 Smart Phone Apps Development
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong