Android Application Development

Overview

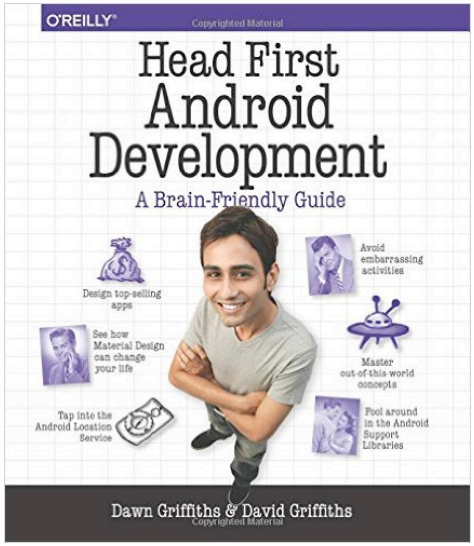
- A quick introduction to Android application development
  - Android system architecture
  - Application development basics
    - Manifest file and resources
    - Activities and intents
    - Life cycle of activities
  - Data persistence
  - Messaging and networking
  - Developing Android services

Development resources

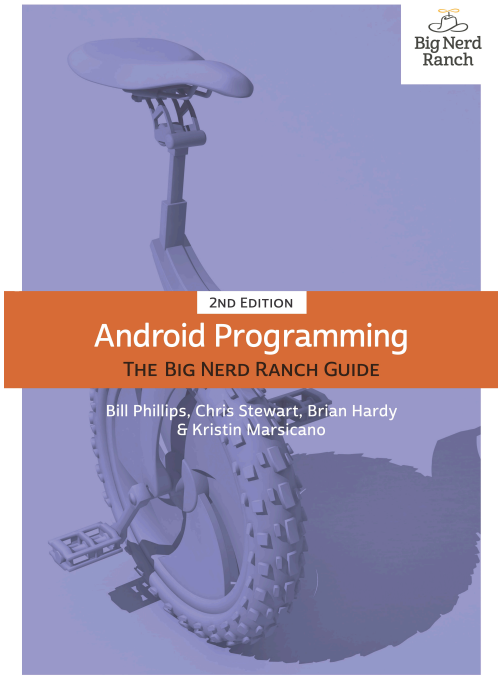
- Developer's Guide
  - http://developer.android.com/guide/index.html
- API Reference
  - http://developer.android.com/reference/packages.html
- Android studio and SDK
  - https://developer.android.com/sdk/index.html
- AVD manager
  - http://developer.android.com/tools/help/avd-manager.html
- Optional:
  - Android phone
  - Device drivers

Recommended reading

- D. Griffiths and D. Griffiths, Head First Android Development



- B. Phillips, C. Steward, B. Hardy and K. Marsicano, Android Programming: The Big Nerd Ranch Guide



Why develop for mobile?

- Multifunction mobile devices have fundamentally changed the way people access the Internet
- Mobile device ownership has surpassed computer ownership in many countries
- The increasing availability of mobile data and WIFI hotspots promoted the ubiquity of mobile devices
- A unique opportunity to build networked applications that are vital to people's lives

Hello Android

- Android is a software platform for an increasingly wide range of network-enabled devices.

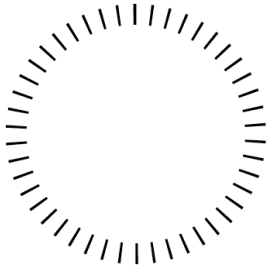
- Android offers native and third-party applications with the same set of APIs
  - Location service, graphics, communication, multimedia, etc.

The anatomy of Android

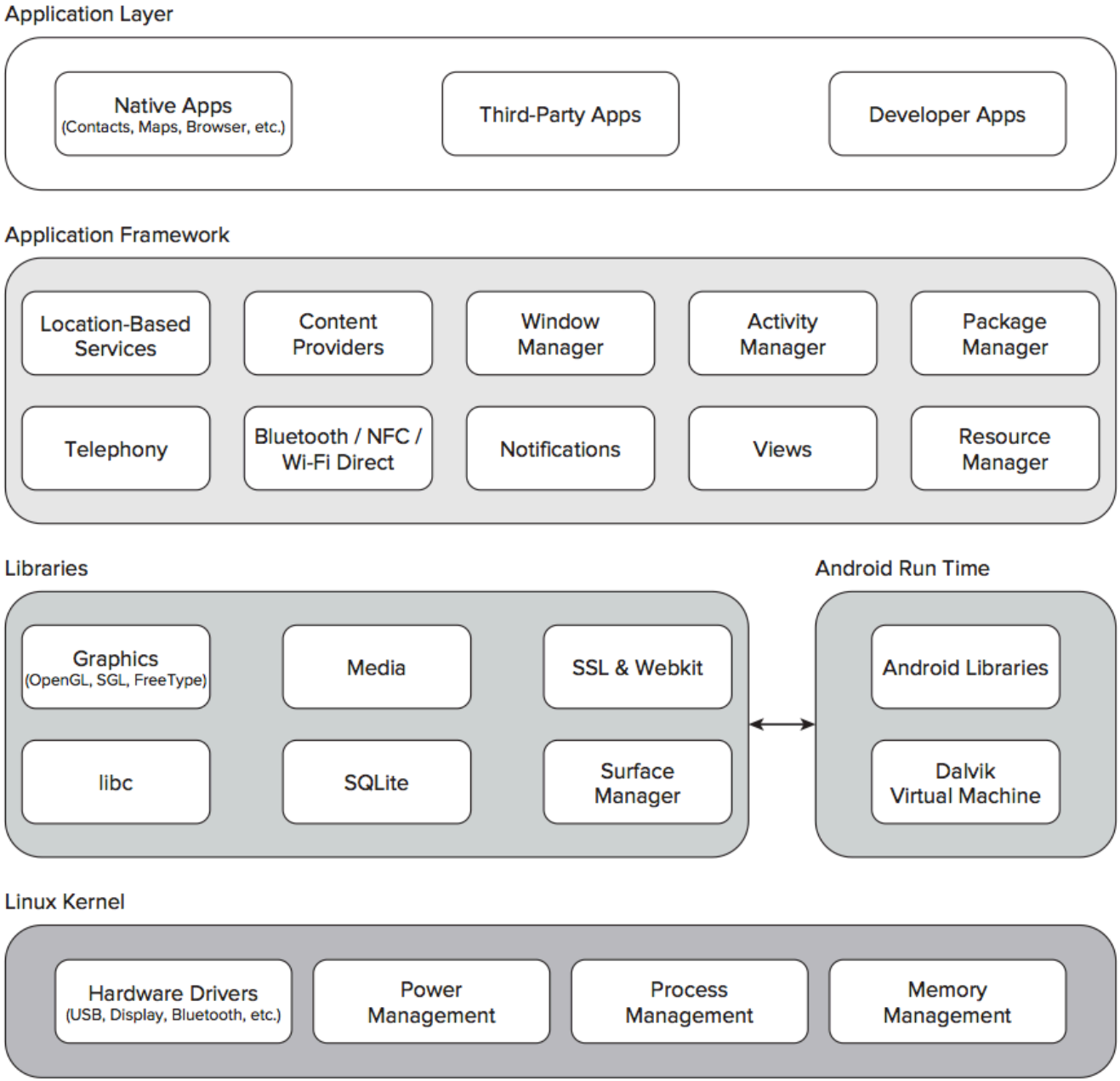
- Android is made up of several interdependent parts:
  - Linux operating system kernel
  - Open-source libraries
  - Dalvik VM for Android applications
  - An application framework
  - A software development kit (SDK)
  - A user interface system
  - A set of core pre-installed applications

Quick exercise

- What are the benefits of using register-based virtual machine?
  - A. Instructions can be executed faster.
  - B. Facilitate re-use of previous execution result.
  - C. Reduce bytecode size.
  - D. Each operand used in an instruction can be addressed implicitly.



Android software stack



Application fundamentals

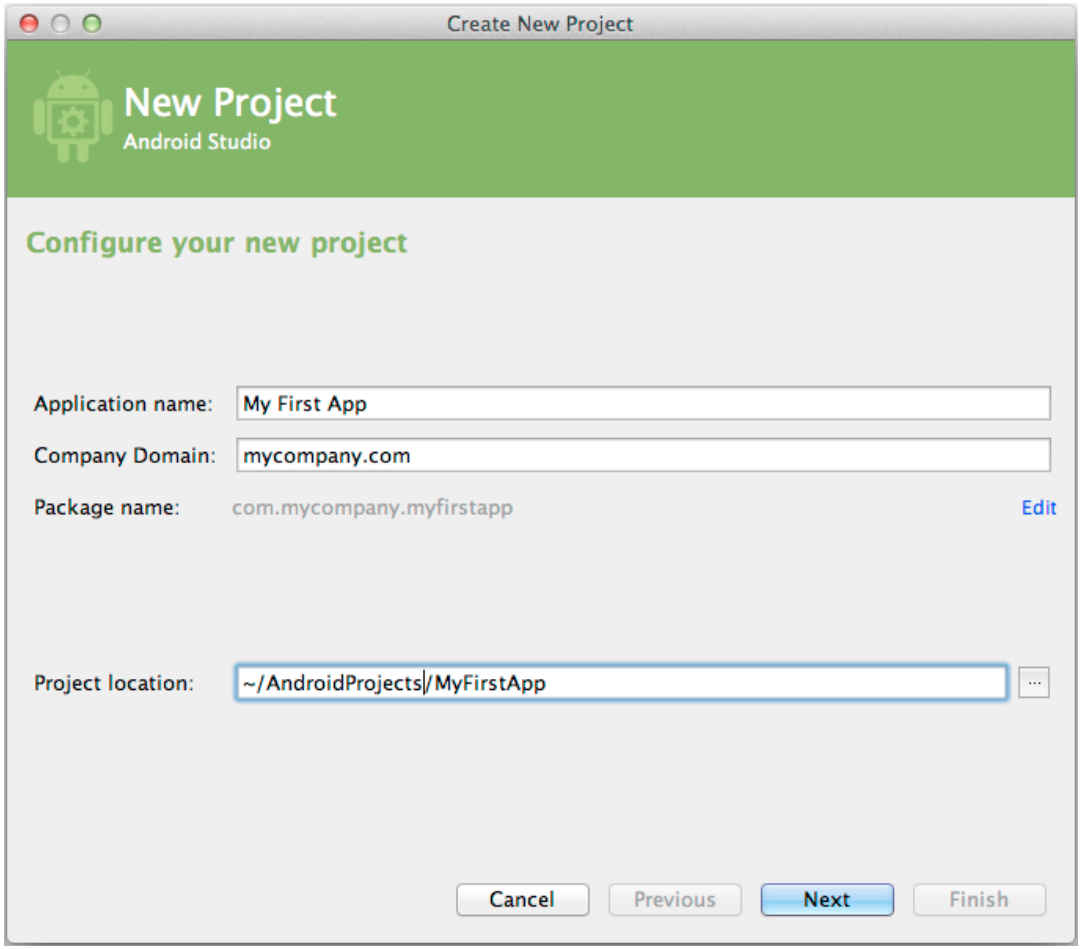
- Apps are written in Java
- Every App runs its own Dalvik VM and Linux process
- Each App is assigned a unique Linux user ID
- Apps can share the same user ID to see each other's files

Types of Android applications

- Foreground
  - e.g. games
- Background
  - e.g. alarm clocks
- Intermittent
  - e.g. email and news applications
- Widgets and live wallpapers

Create a simple application

- Tutorials on building your first app
  - https://developer.android.com/training/basics/firstapp/index.html



The application manifest file

- Each Android project includes a manifest file
  - defines the structure and metadata of your application, its components, and its requirements.
- Overall structure

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.paad.myapp"
          android:versionCode="1"
          android:versionName="0.9 Beta"
          android:installLocation="preferExternal">
          [ ... manifest nodes ... ]
</manifest>
```

Sub-nodes to be included here

Some sub-nodes in a Manifest file

- The uses-sdk element

```
<uses-sdk android:minSdkVersion="6"
          android:targetSdkVersion="15"/>
```

- The uses-feature element

```
<uses-feature android:name="android.hardware.nfc" />
```

- The uses-permission element

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

- The application element
  - A manifest can contain only one application node
  - specify the metadata for your application
  - acts as a container for other application components

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.messenger" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".CreateMessageActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".ReceiveMessageActivity"
            android:label="@string/title_activity_receive_message" >
        </activity>

    </application>

</manifest>
```

This is the package name we specified.

This is the first activity, Create Message Activity.

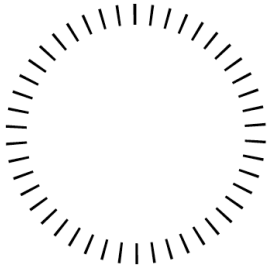
This bit specifies that it's the main activity of the app.

This says the activity can be used to launch the app.

This is the second activity, Receive Message Activity.

Android Studio added these lines for us when we added the second activity.

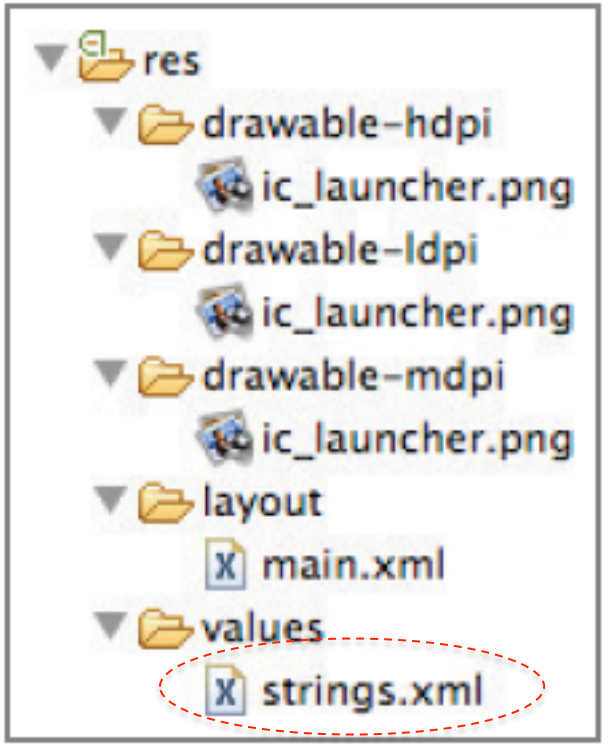
Quick exercise



- Which task cannot be performed by using the manifest file?
  - A. Defines components of an application
  - B. Protects application from accessing any protected parts of the system
  - C. Lists the instrumentation classes.
  - D. Specifies the user ID to be used by the Android application.

Externalizing resources

- It's always good practice to keep non-code resources external to your code.
- Android dynamically selects resources from resource trees that contain different values for alternative hardware configurations, languages, and locations.
- R class file is automatically generated to enable resource reference in code.



Define string resources – an example

- Define a string in strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

- Use the defined string.

```
Resources myResources = getResources();

CharSequence styledText = myResources.getText(R.string.stop_message);
Drawable icon = myResources.getDrawable(R.drawable.app_icon);

int opaqueBlue = myResources.getColor(R.color.opaque_blue);
```

```
String string =  getString (R.string.hello);
```

About asset

- Assets are like a little file system that ships with your packaged application.
  - commonly used for loading graphics and sound
- Major steps
  - Import assets by using Android Studio
  - Create reference to asset
  - Access asset

```
// AssetFileDescriptors are different from FileDescriptors,
AssetFileDescriptor assetFd = mAssets.openFd(assetPath);
// but you get can a regular FileDescriptor easily if you need to.
FileDescriptor fd = assetFd.getFileDescriptor();
```

Using resources

- Using resources in code
  - Access resources in code using the static R class
  - Each of the subclasses within R exposes its associated resources as variables

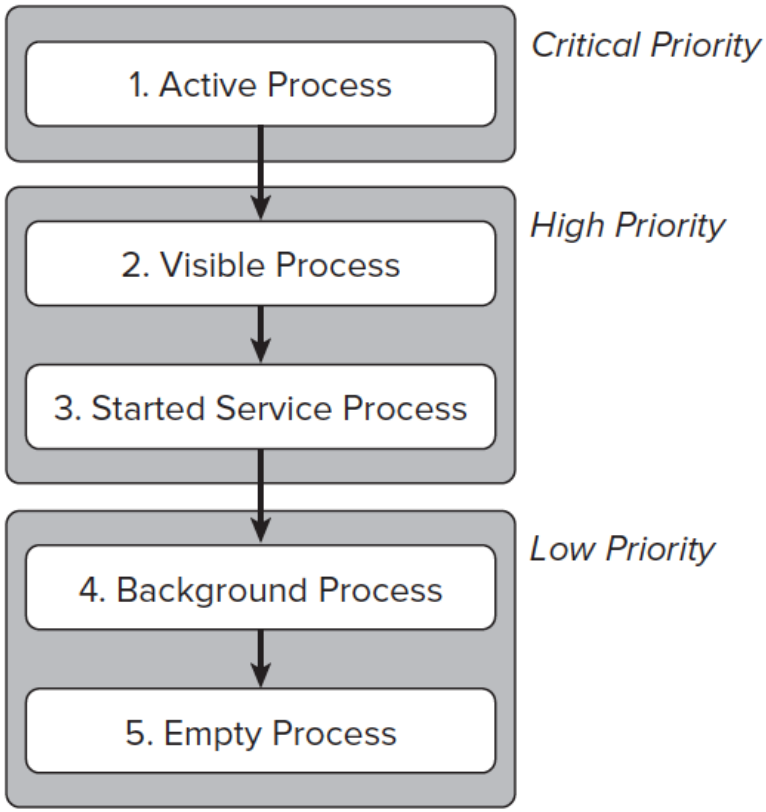
```
Resources myResources = getResources();

CharSequence styledText = myResources.getText(R.string.stop_message);
Drawable icon = myResources.getDrawable(R.drawable.app_icon);

int opaqueBlue = myResources.getColor(R.color.opaque_blue);
```

- Referencing resources within resources

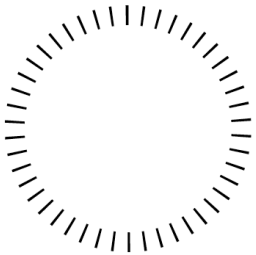
```
<EditText
  android:id="@+id/myEditText"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/stop_message"
  android:textColor="@color/opaque_blue"
/>
```

Android application lifecycle

- Android applications have limited control over their own lifecycle.
- Application priority
  - Equals to its highest-priority component.
- All Android applications continue running and in memory until the system needs resources for other applications.



Quick exercise



- Two applications A and B have the same priority.
- A spends longer time staying in that priority level than B.
- B depends on a content provider supplied by A.

- Which application might be killed the first and why?

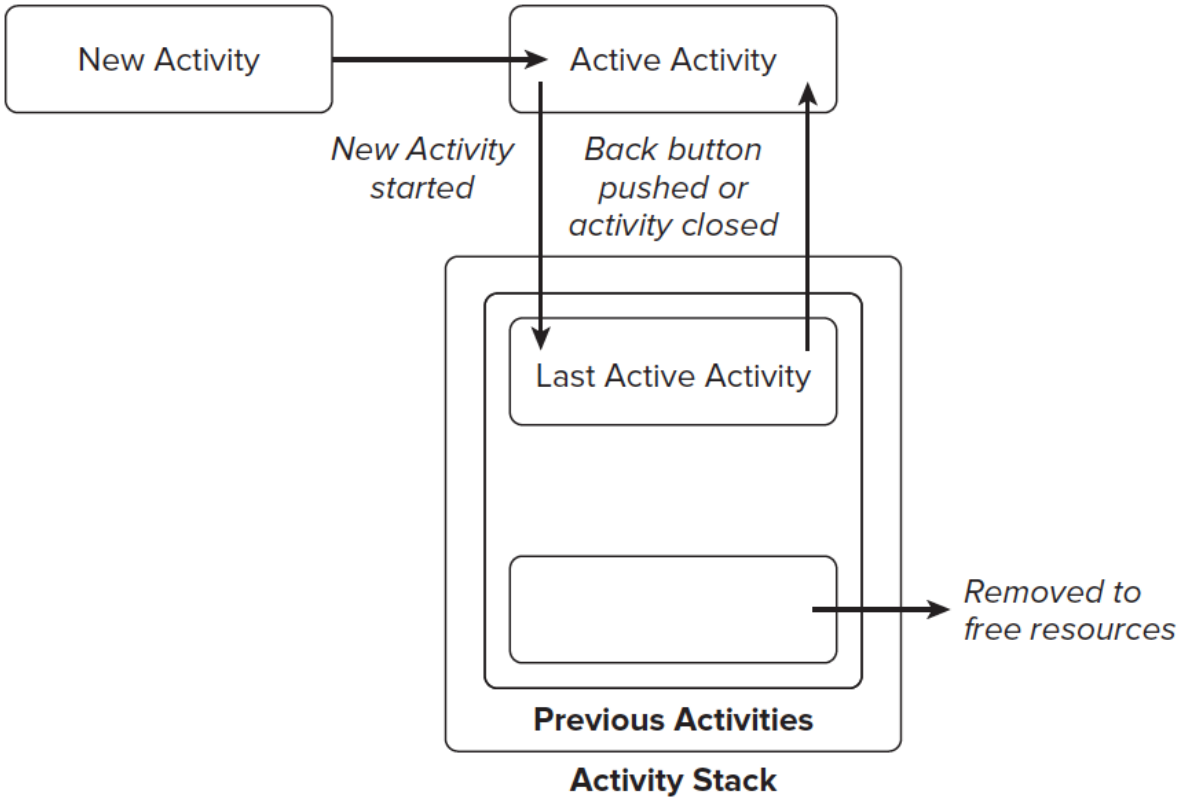
What is an activity?

- An activity is a window that contains the user interface of your application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.Activities"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="9" />
</manifest>
```

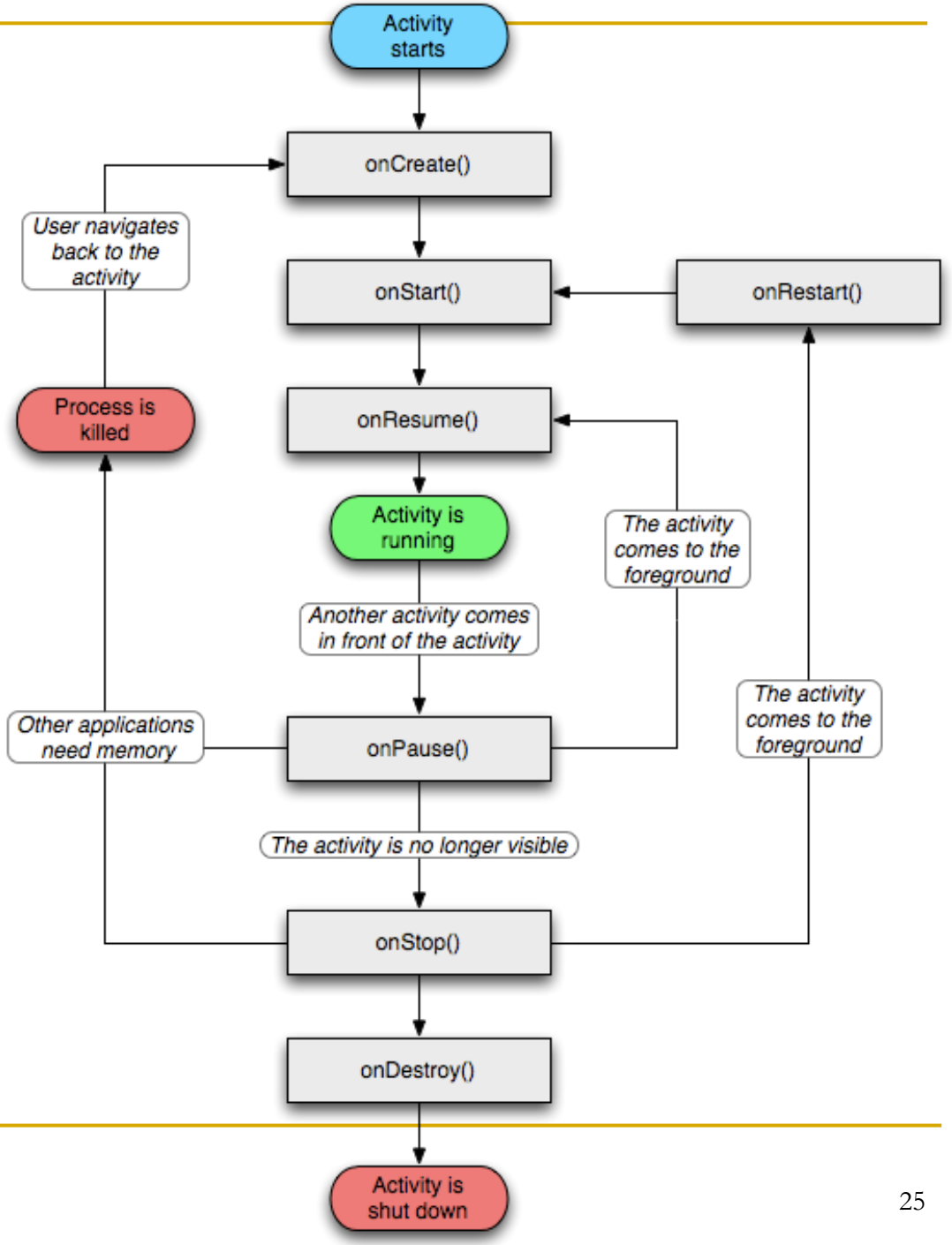
Activity lifecycle

- Activity's state helps determine the priority of its parent application.
- Activity stack
  - The state of each Activity is determined by its position on the Activity stack



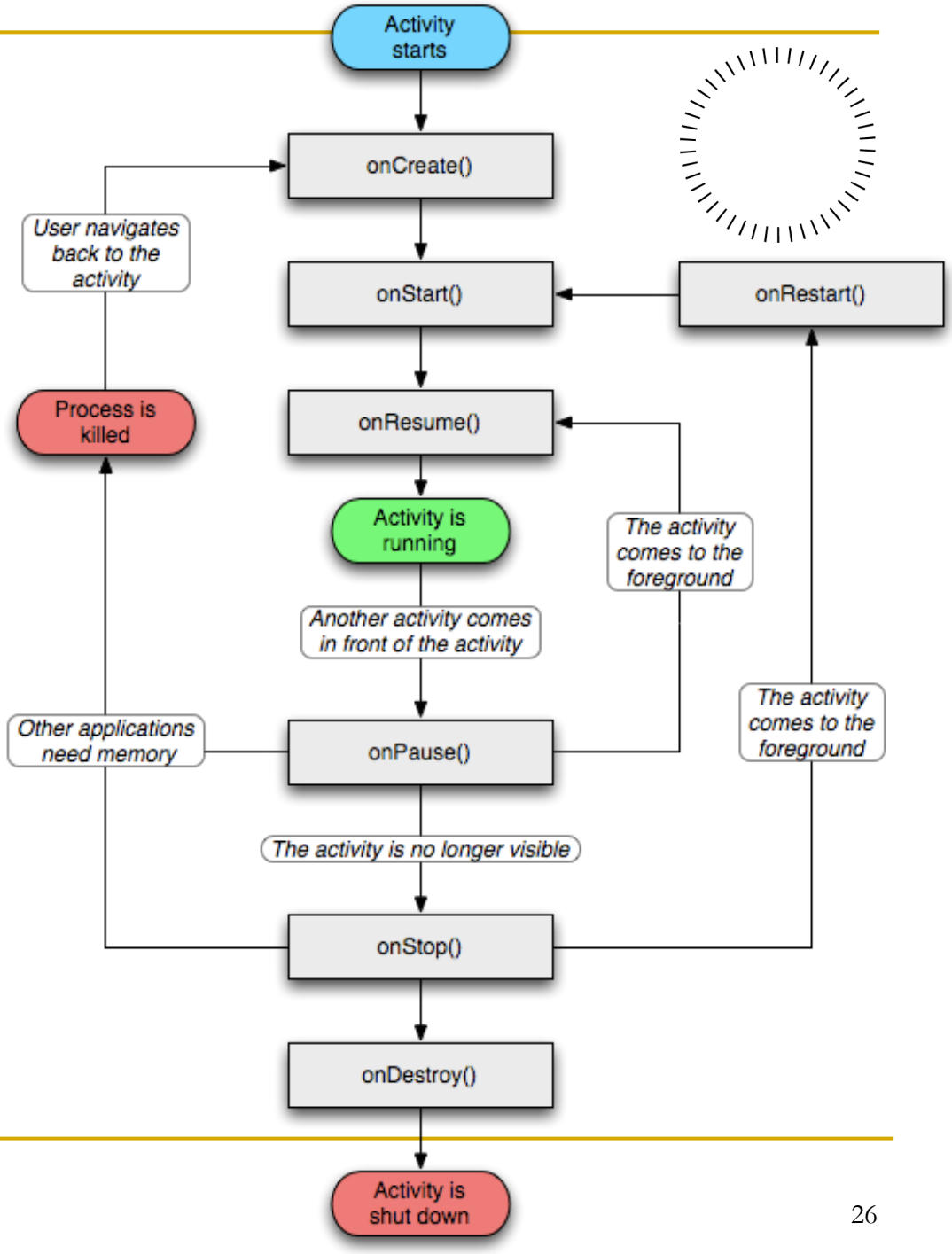
Activity state

- Active state
- Paused state
- Stopped state
- Inactive state



Quick exercise

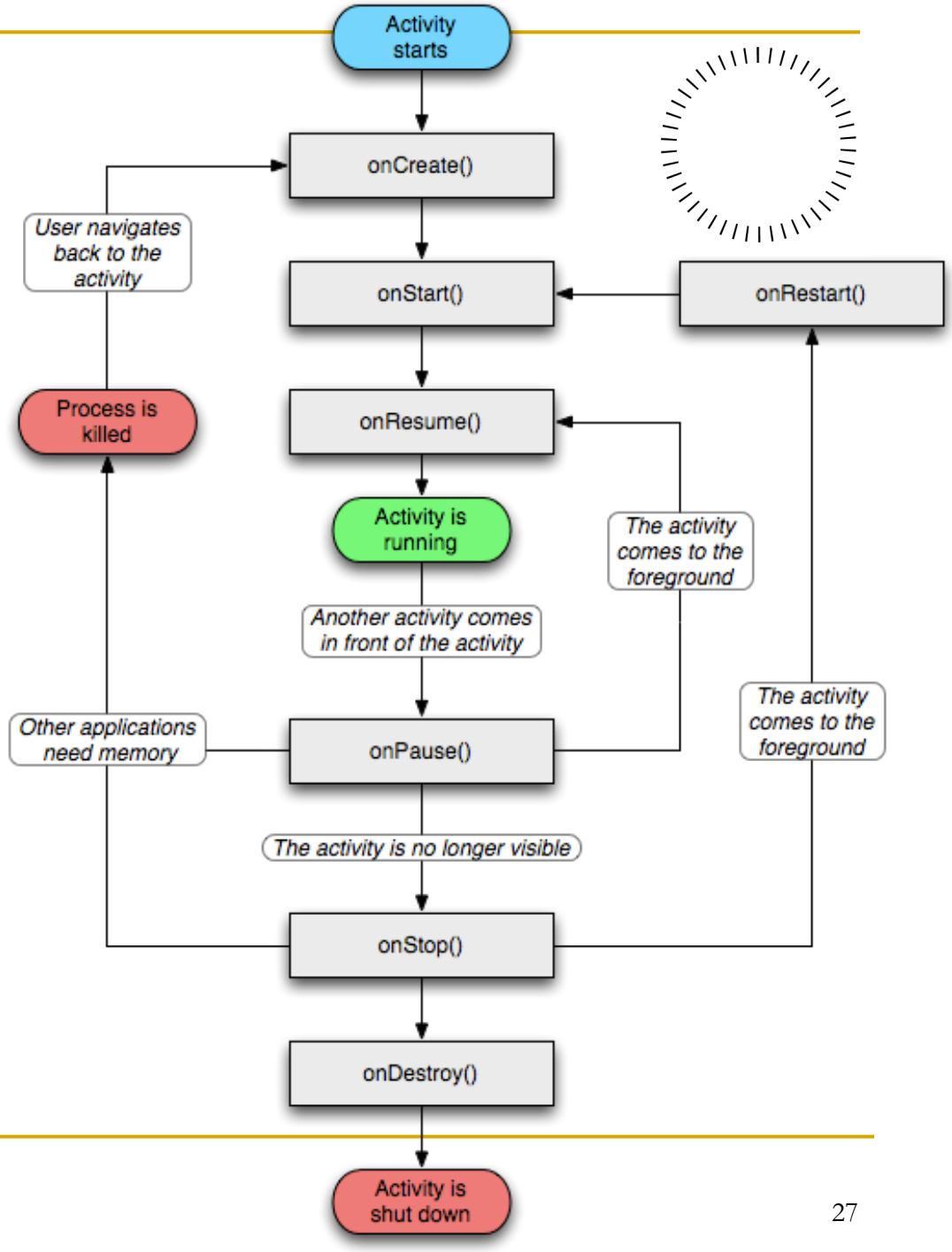
- Which event handler should be used to make state information persistent?
  - A. onRestart()
  - B. onStart()
  - C. onStop()
  - D. onResume()



Quick exercise

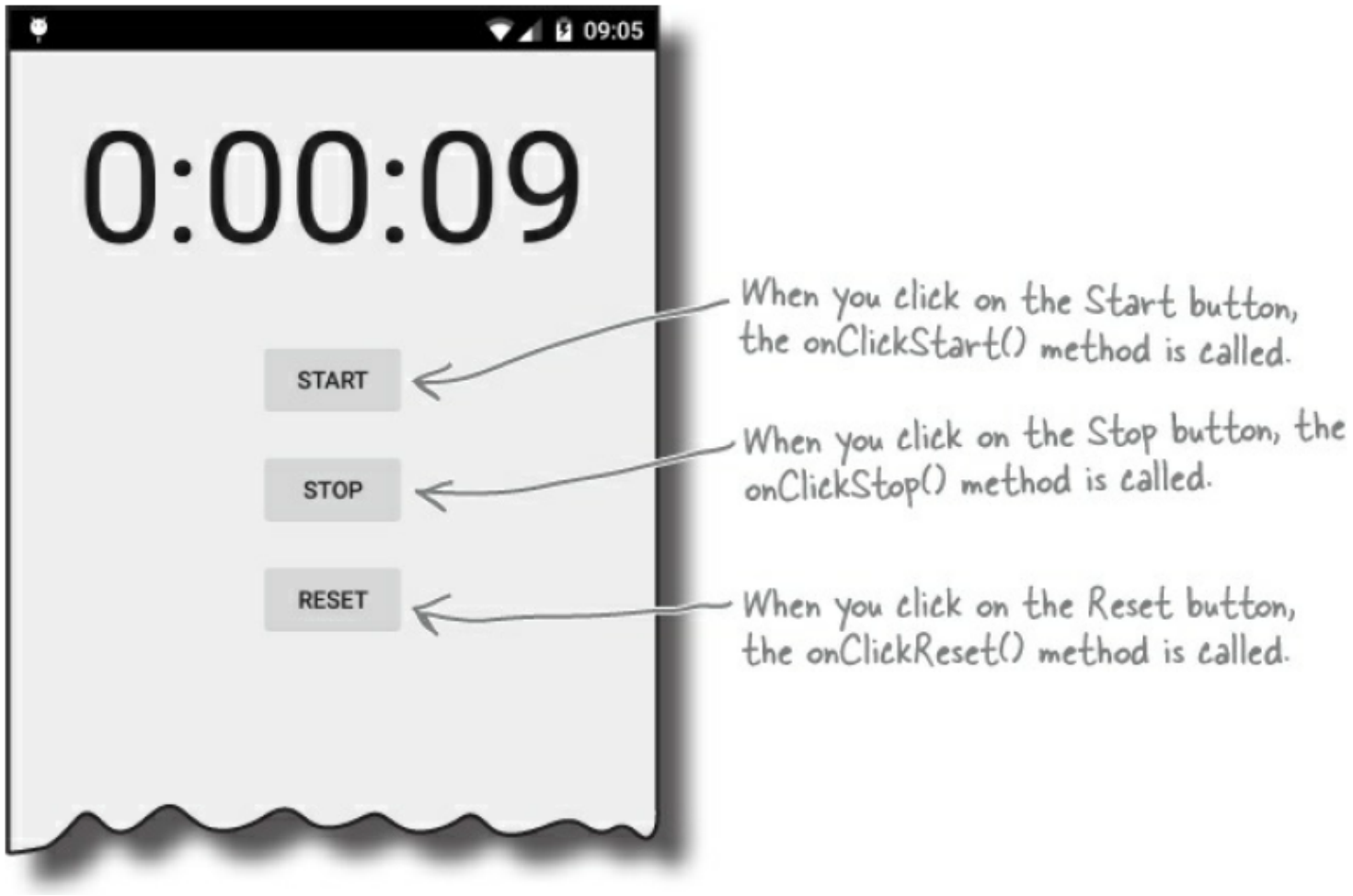
- What event handlers will definitely be called when an activity (and its UI) is pushed to the background and later made visible again?

  - A. onCreate()
  - B. onRestart()
  - C. onStart()
  - D. onStop()
  - E. onResume()



Case study – working with activity life cycle

- Develop a stop watch app.



Java code

```
public class StopwatchActivity extends Activity {

    private int seconds = 0;
    private boolean running;

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


    //Start the stopwatch running when the Start button is clicked.
    public void onClickStart(View view) {
        running = true;
    }

    //Stop the stopwatch running when the Stop button is clicked.
    public void onClickStop(View view) {
        running = false;
    }

    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }
}
```

Use seconds and running to record the number of seconds passed and whether the stopwatch is running.

This gets called when the Start button is clicked.

Start the stopwatch running.

This gets called when the Stop button is clicked.

Stop the stopwatch running.

This gets called when the Reset button is clicked.

Stop the stopwatch running and set the seconds to 0.

Java code

```
public class StopwatchActivity extends Activity {

    private int seconds = 0;
    private boolean running;

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

Use seconds and running to record the number of seconds passed and whether the stopwatch is running.

```
<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/time_view"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:onClick="onClickStart"
    android:text="@string/start" />
```

This is for the Start button. It calls a method called onClickStart() when it gets clicked.

```
    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }
}
```

Stop the stopwatch running and set the seconds to 0.

This gets called when the Reset button is clicked.

Controlling repeated update to clock counter

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format("%d:%02d:%02d",
                    hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
```

Create a new Handler.

Call the post() method, passing in a new Runnable. The post() method processes codes without a delay, so the code in the Runnable will run almost immediately.

The Runnable run() method contains the code you want to be run—in our case, the code to update the text view.

Post the code in the Runnable to be run again after a delay of 1,000 milliseconds, or 1 second. As this line of code is included in the Runnable run() method, this will keep getting called.

Controlling repeated update to clock counter

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            handler.postDelayed(this, 1000);
        }
    });
}
```

Create a new Handler.

Call the post() method, passing in a new Runnable. The post() method processes codes without a delay, so the code in the Runnable will run almost immediately.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stopwatch);
    runTimer();
}
```

We're using a separate method to update the stopwatch. We're starting it when the activity is created.

Post the code in the Runnable to be run again after a delay of 1,000 milliseconds, or 1 second. As this line of code is included in the Runnable run() method, this will keep getting called.

Test our stop watch app

Keep the watch going after orientation change

- Keep state info before an activity is destroyed.

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
}
```

Save the values of the seconds and running

- Restore preserved state upon creation

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stopwatch);
    if (savedInstanceState != null) {
        seconds = savedInstanceState.getInt("seconds");
        running = savedInstanceState.getBoolean("running");
    }
    runTimer();
}
```

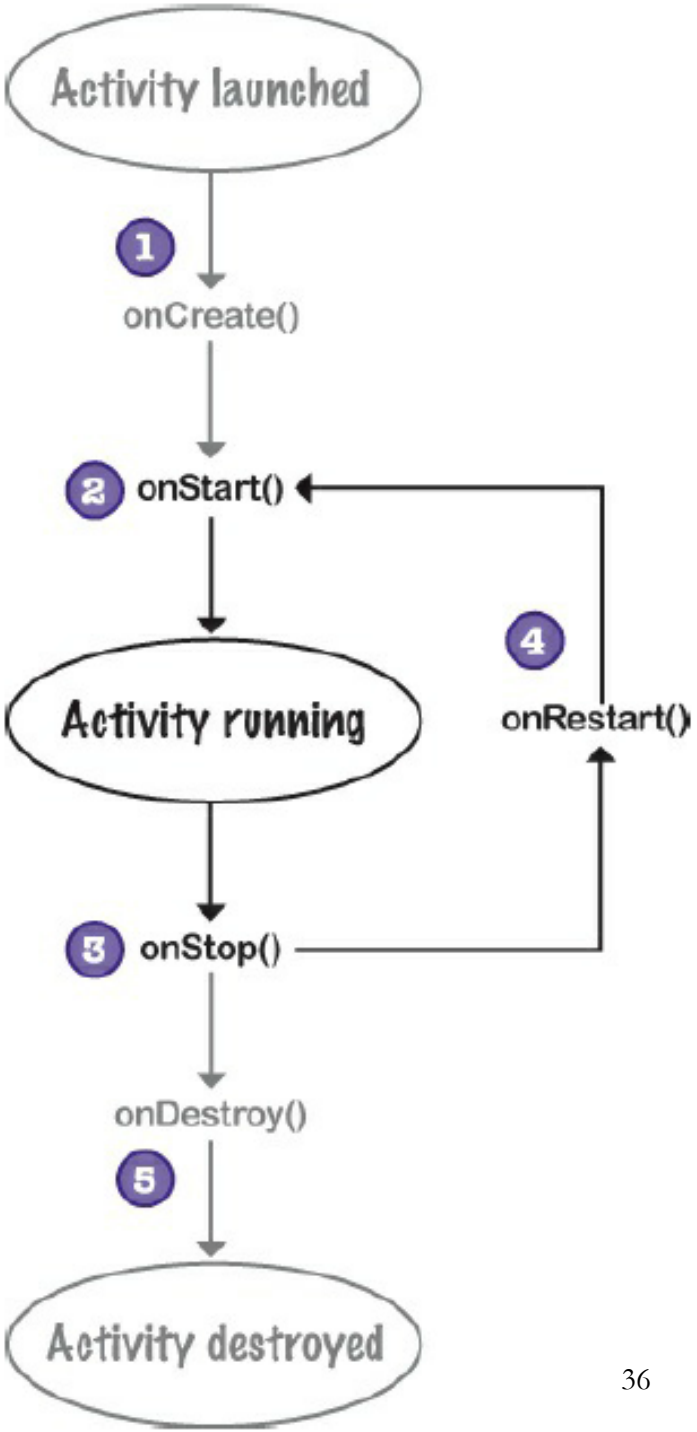
Retrieve the values of the seconds and running variables from the Bundle.

Questions to ponder

- Why does Android want to re-create an activity just because I rotated the screen?
- Why doesn't Android store every instance variable automatically? Why do I have to write all of that code myself?
- How can we stop the watch when the app is no longer in the foreground?

Capture the activity state change

- Event handlers to be called by Android when an activity switches between foreground and background



Adding new attribute

- Introduce new attribute

```
private int seconds = 0;
private boolean running;
private boolean wasRunning;
```

A new variable, wasRunning, records whether the stopwatch was running before the onStop() method was called so that we know whether to set it running again when the activity becomes visible again.

- Preserving new attribute

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}
```

Save the state of the wasRunning variable.

```
wasRunning = savedInstanceState.getBoolean("wasRunning");
```

Restore the state of the wasRunning variable if the activity is re-created.

Stop the watch in the background

```
@Override
protected void onStop() {
    super.onStop();
    wasRunning = running;
    running = false;
}
```

Record whether the stopwatch was running when the onStop() method was called.

```
@Override
protected void onStart() {
    super.onStart();
    if (wasRunning) {
        running = true;
    }
}
```

Implement the onStart() method. If the stopwatch was running, set it running again.

Important things to remember

- Only the main thread can update the user interface.
- A device configuration change results in the activity being destroyed and re-created.
- Your activity inherits the lifecycle methods from the Android Activity class.
  - If you override any of these methods, you need to call up to the method in the superclass.

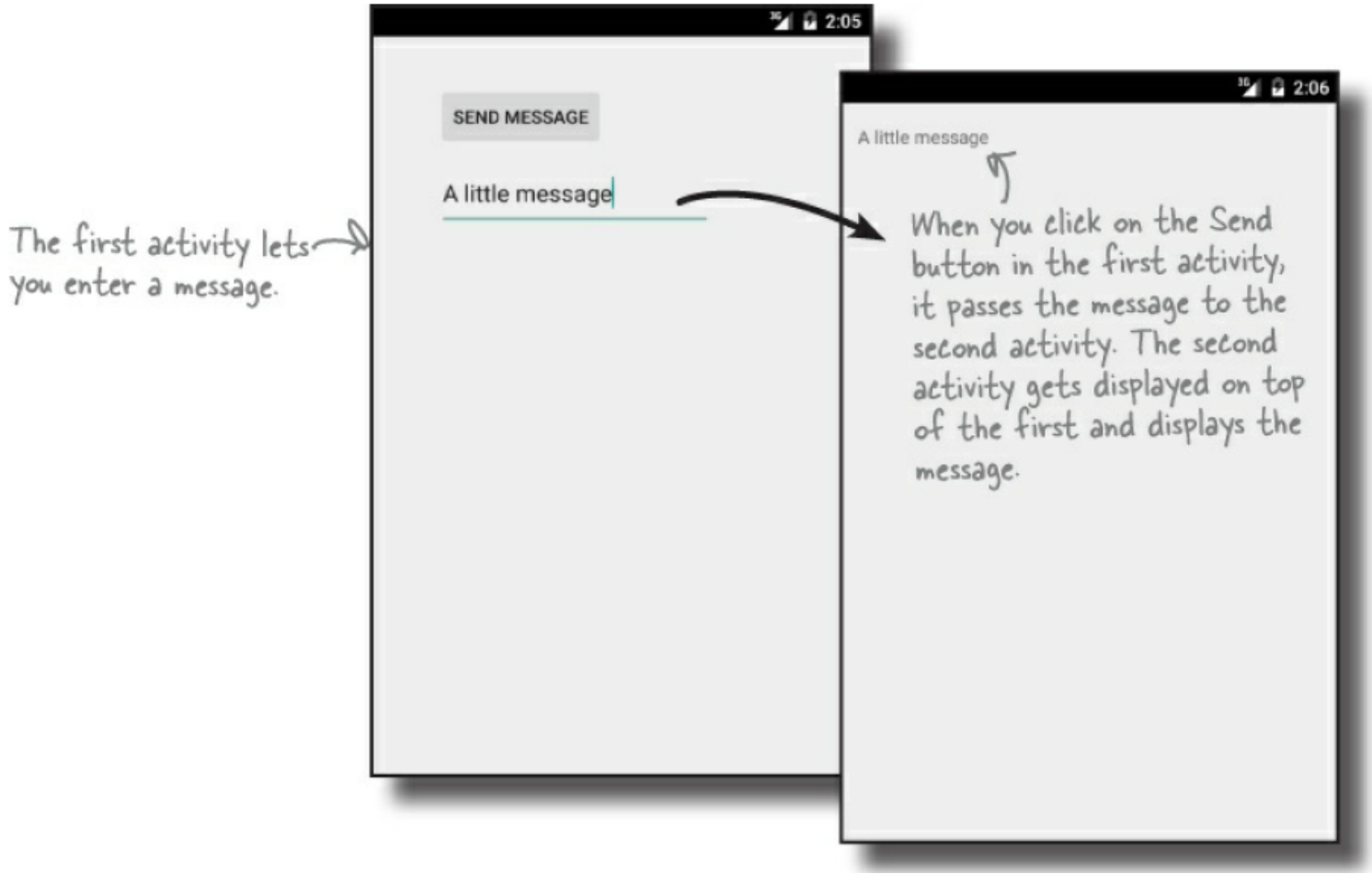
Quick exercise

- What methods will be called in sequence if user starts the activity, starts using it, then switches to another app?
  - A. onCreate()
  - B. onPause()
  - C. onRestart()
  - D. onResume()
  - E. onStop()
  - F. onStart()
  - G. onDestroy()

Intent

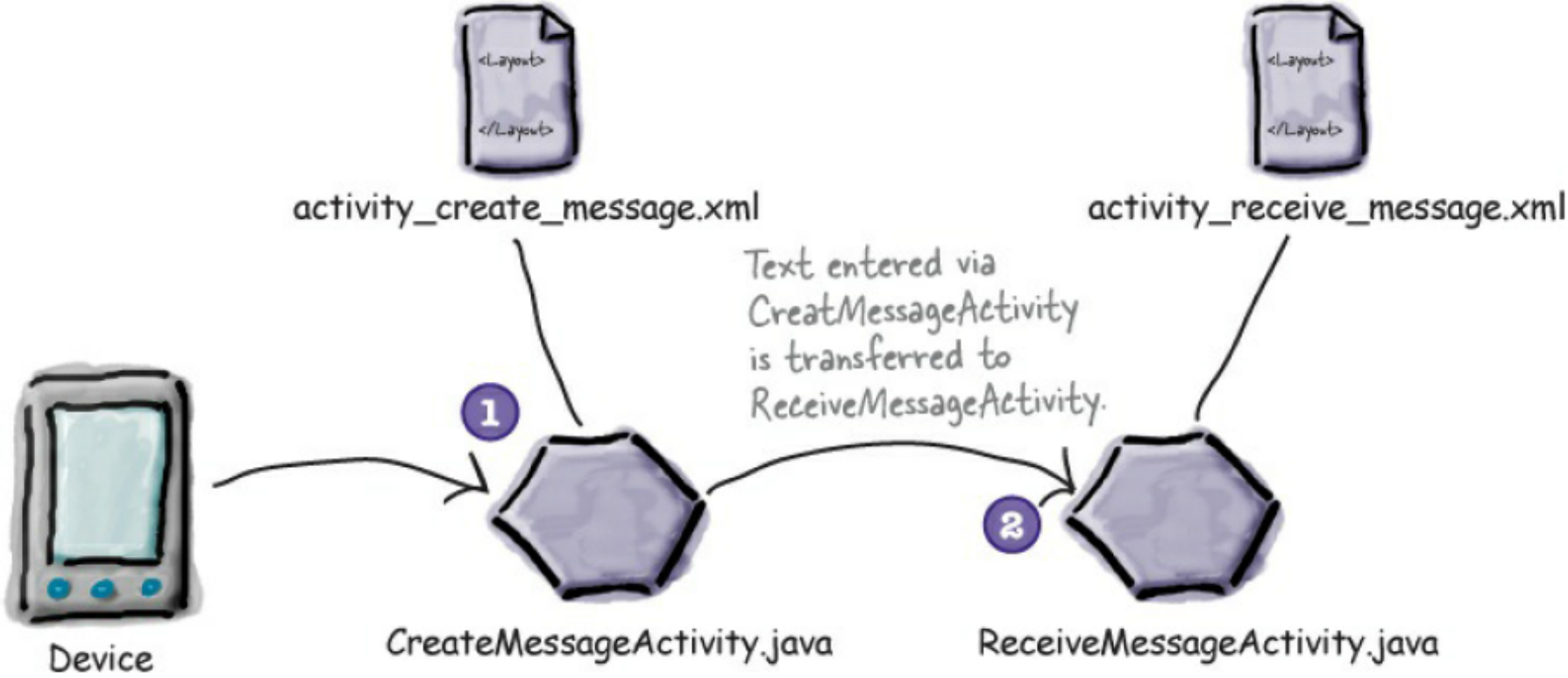
- Intents support message-passing within or across applications.
- Main usage
  - Explicitly start a particular Service or Activity using its full class name
  - Start an Activity or Service to perform an action with (or on) a particular piece of data
  - Broadcast that an event has occurred

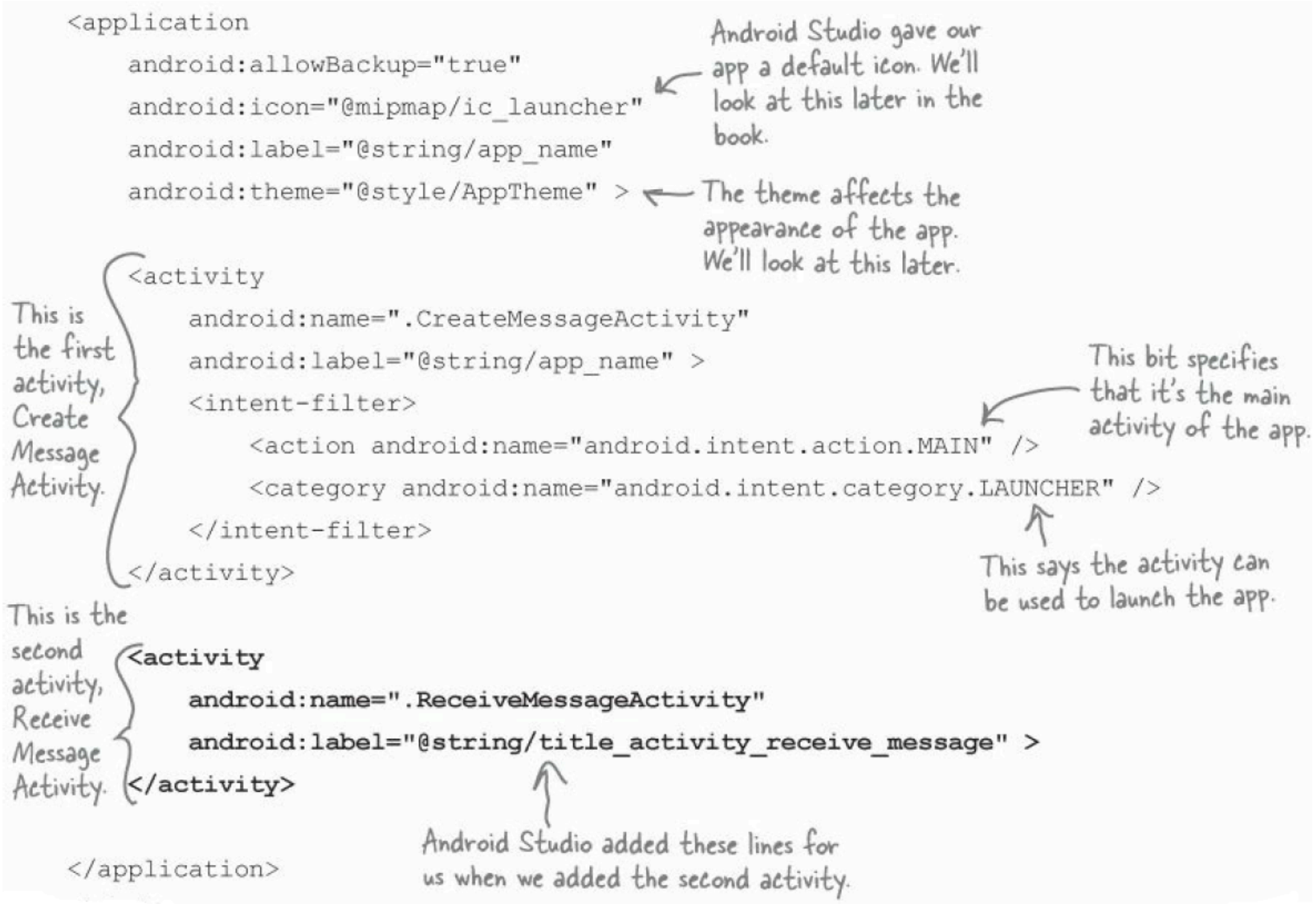
Case study – activity and intents



Project structure

- An app with two activities.



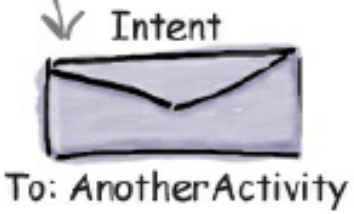

<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
Android Studio gave our app a default icon. We'll look at this later in the book.
The theme affects the appearance of the app. We'll look at this later.
This is the first activity, Create Message Activity.
<activity
android:name=".CreateMessageActivity"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
This bit specifies that it's the main activity of the app.
This says the activity can be used to launch the app.
This is the second activity, Receive Message Activity.
<activity
android:name=".ReceiveMessageActivity"
android:label="@string/title_activity_receive_message" >
</activity>
Android Studio added these lines for us when we added the second activity.
</application>

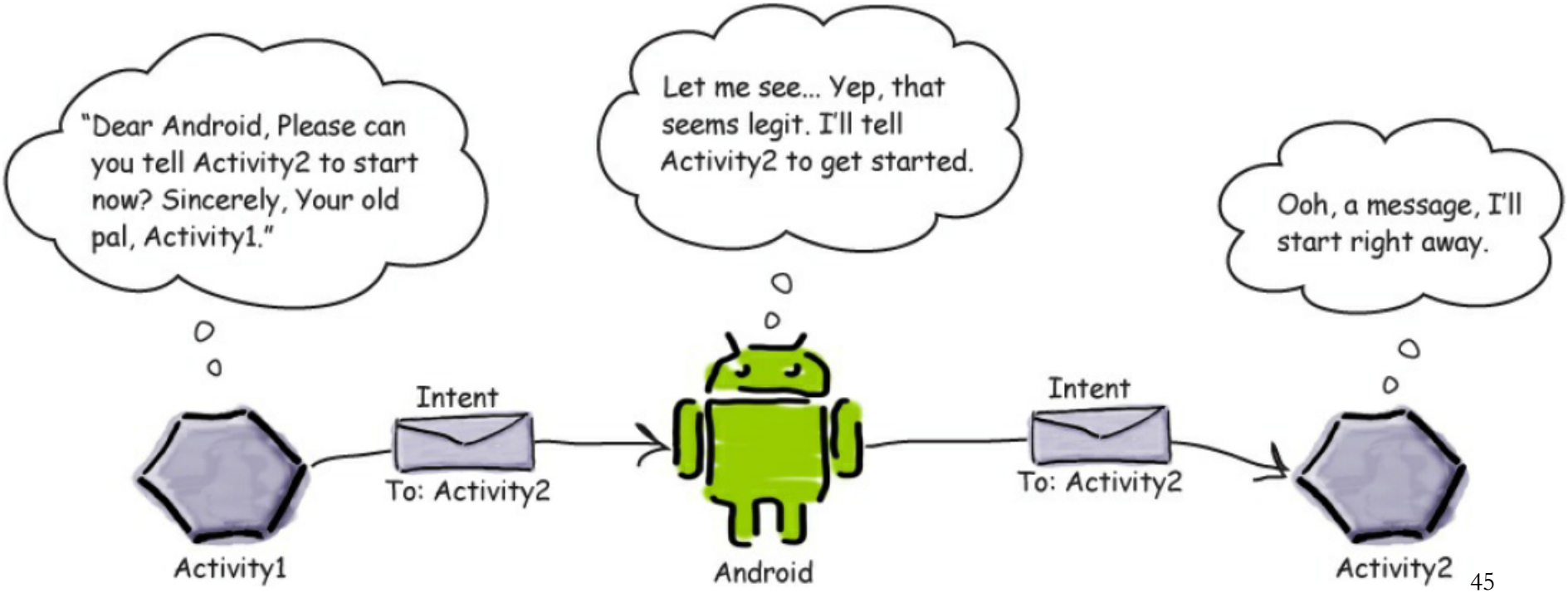
Use intent to chain activities together

```
Intent intent = new Intent(this, Target.class);

startActivity(intent);
```

The intent specifies the activity you want to receive it. It's like putting an address on an envelope.

startActivity() starts the activity specified in the intent.



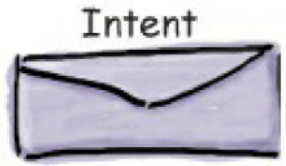


Pass data through intent

- Add information to an intent

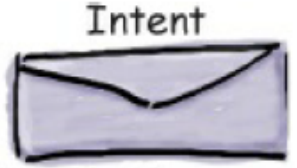
putExtra() lets you put extra information in the message you're sending.

```
intent.putExtra("message", value);
```



To: ReceiveMessageActivity
message: "Hello!"

- Retrieve information from an intent



To: ReceiveMessageActivity
message: "Hello!"

```
Intent intent = getIntent();
String string = intent.getStringExtra("message");
```

Get the intent.

Get the string passed along with the intent that has a name of "message".

Complete coding

```
public class CreateMessageActivity extends Activity {

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

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        ..........
        ..........
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        ..........
        startActivity(intent);
    }
}
```

Answer

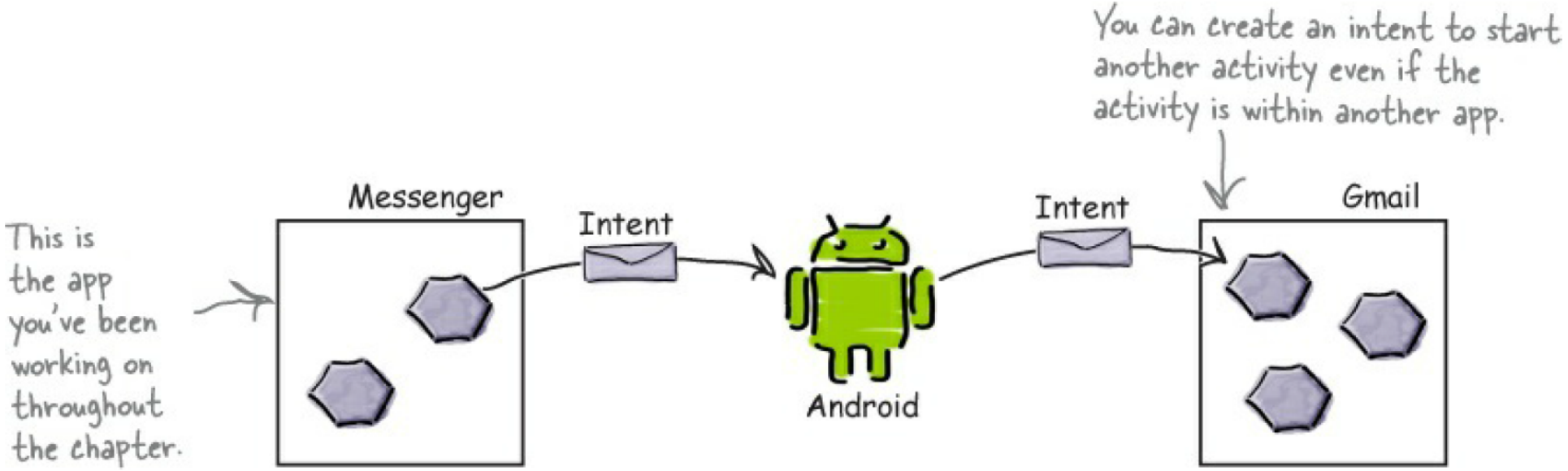
```
//Call onSendMessage() when the button is clicked
public void onSendMessage(View view) {
    EditText messageView = (EditText) findViewById(R.id.message);
    String messageText = messageView.getText().toString();
    Intent intent = new Intent(this, ReceiveMessageActivity.class);
    intent.putExtra("message", messageText);
    startActivity(intent);
}
```

Get the text from the editable text field with an ID of message.

Add the text to the intent, giving it a name of "message".

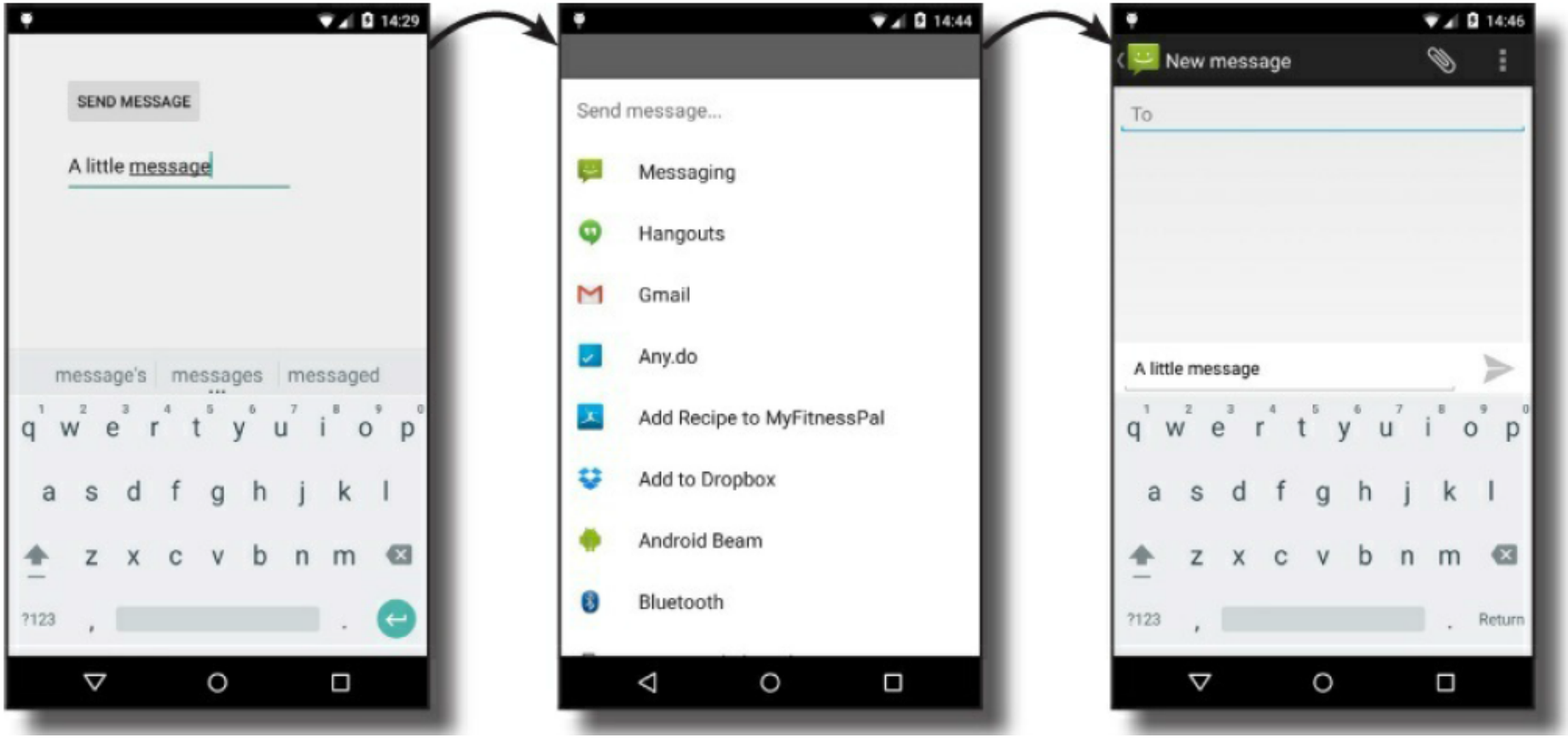
Activate third-party activities

- Intent can start activity in other apps.



  - Create an intent that specifies an action.

Example – send message by email



Things to do …

- Create implicit intent

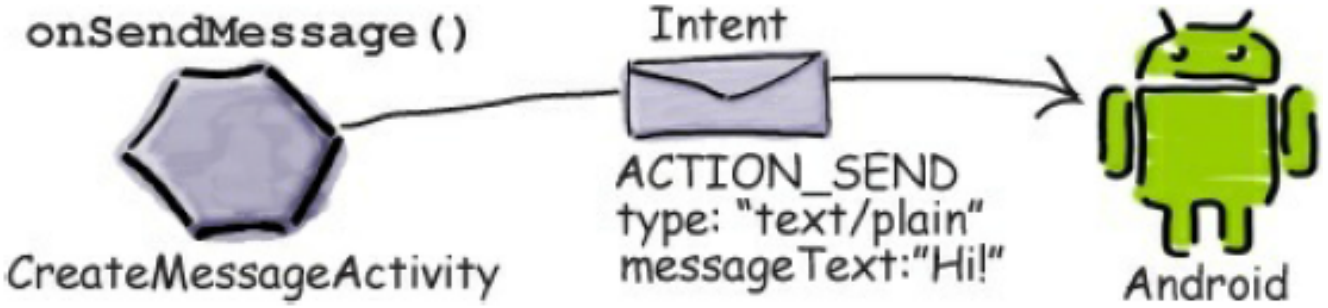
```
Intent intent = new Intent(Intent.ACTION_SEND);
```

- Add extra information

```
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, messageText);
```

These attributes relate to Intent.ACTION_SEND. They're not relevant for all actions.

- Pass intent to Android



Things to do …

- Create implicit intent

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

- Add extra information

```
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, messageText);
```

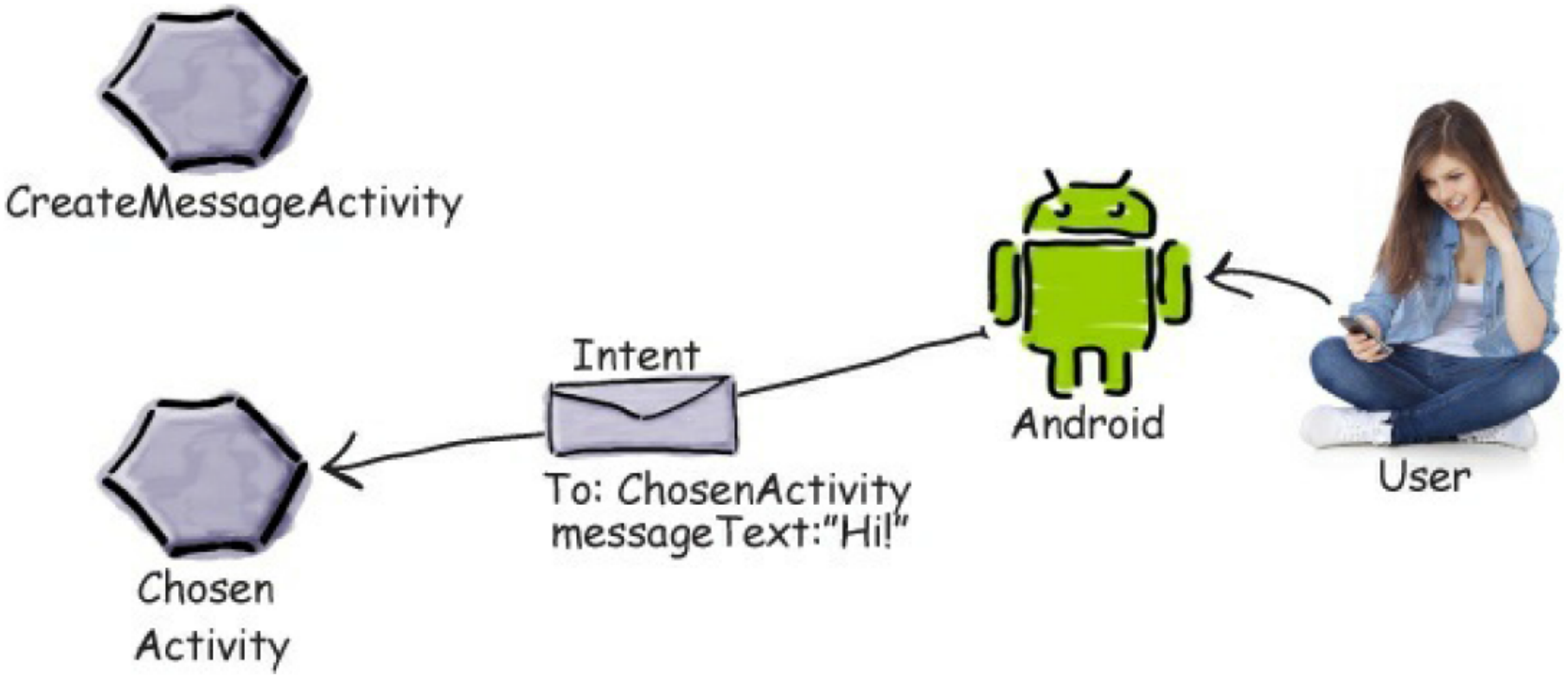
These attributes relate to Intent.ACTION_SEND. They're not relevant for all actions.

- Pass intent to Android

```
PackageManager packageManager = getActivity().getPackageManager();
if (packageManager.resolveActivity(pickContact,
        PackageManager.MATCH_DEFAULT_ONLY) == null) {
    mSuspectButton.setEnabled(false);
}
```

Fire an implicit intent

- User chooses an activity



Intent filters

- The intent filter tells Android which activities can handle which actions

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

This tells Android the activity can handle ACTION_SEND.

The intent filter must include a category of DEFAULT or it won't be able to receive implicit intents.

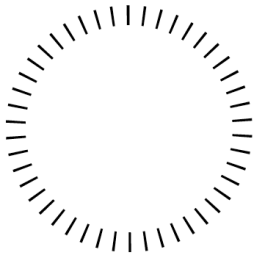
These are the types of data the activity can handle.

Here's the intent.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

- Which activity can be used to send a plaintext message ?

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="*/*"/>
    </intent-filter>
</activity>
```



```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.MAIN"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.MAIN"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

More questions …

- Should I use implicit or explicit intents?
- An activity's intent filter can specify a category as well as an action. What's the difference between the two?
- What happens when there is no activity that can perform a requested action in an implicit intent?

Returning results from activities

- Launch sub-activities

```
private static final int SHOW_SUBACTIVITY = 1;

private void startSubActivity() {
  Intent intent = new Intent(this, MyOtherActivity.class);
  startActivityForResult(intent, SHOW_SUBACTIVITY);
}
```

- Generate a result in a sub-activity

```
okButton.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    long selected_horse_id = listView.getSelectedItemId();

    Uri selectedHorse = Uri.parse("content://horses/" +
                                  selected_horse_id);
    Intent result = new Intent(Intent.ACTION_PICK, selectedHorse);

    setResult(RESULT_OK, result);
    finish();
  }
```

Handling sub-activity results

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

  super.onActivityResult(requestCode, resultCode, data);

  switch(requestCode) {
    case (SELECT_HORSE):
      if (resultCode == Activity.RESULT_OK)
        selectedHorse = data.getData();
      break;

    case (SELECT_GUN):
      if (resultCode == Activity.RESULT_OK)
        selectedGun = data.getData();
      break;

    default: break;
  }
}
```

Data persistence

- Save simple data using the SharedPreferences object
- Traditional file system
- SQLite database

Saving data using SharedPreferences

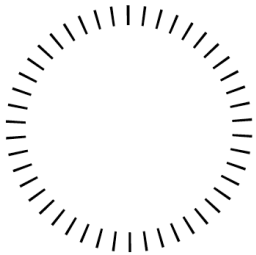
```
btn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //---get the SharedPreferences object---
        prefs = getSharedPreferences(prefName, MODE_PRIVATE);
        SharedPreferences.Editor editor = prefs.edit();

        //---save the values in the EditText view to preferences---
        editor.putFloat(FONT_SIZE_KEY, editText.getTextSize());
        editor.putString(TEXT_VALUE_KEY, editText.getText().toString());

        //---saves the values---
        editor.commit();

        //---display file saved message---
        Toast.makeText(getBaseContext(),
            "Font size saved successfully!",
            Toast.LENGTH_SHORT).show();
    }
});
```

Loading data from SharedPreferences

```
//---load the SharedPreferences object---
SharedPreferences prefs = getSharedPreferences(prefName, MODE_PRIVATE);

//---set the TextView font size to the previously saved values---
float fontSize = prefs.getFloat(FONT_SIZE_KEY, 12);
```

Quick exercise

- When should we use SharedPreferences?
  - A. Store a large quantity of data
  - B. Store simple unstructured data
  - C. Store structured data
  - D. Store shared data



Saving data to files in internal storage

- Open a file output stream

```
FileOutputStream fOut =
    openFileOutput("textfile.txt",
    MODE_WORLD_READABLE);
```

- Convert byte stream into character stream

```
OutputStreamWriter osw = new
    OutputStreamWriter(fOut);
```

- Write text to a file

```
osw.write(str);
osw.flush();
osw.close();
```

Reading data from a file

- Open the input channel

```
FileInputStream fIn =
    openFileInput("textfile.txt");
InputStreamReader isr = new
    InputStreamReader(fIn);
```

- Proceed to read data in

```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";

int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{
    //---convert the chars to a String---
    String readString =
        String.copyValueOf(inputBuffer, 0,
        charRead);
    s += readString;

   inputBuffer = new char[READ_BLOCK_SIZE];
}
```

Accessing an external storage

- Code snippet

```
//---SD Card Storage---
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +
    "/MyFiles");
directory.mkdirs();
File file = new File(directory, "textfile.txt");
FileOutputStream fOut = new FileOutputStream(file);

OutputStreamWriter osw = new
OutputStreamWriter(fOut);

//---write the string to the file---
osw.write(str);
osw.flush();
```

Using Internet resources

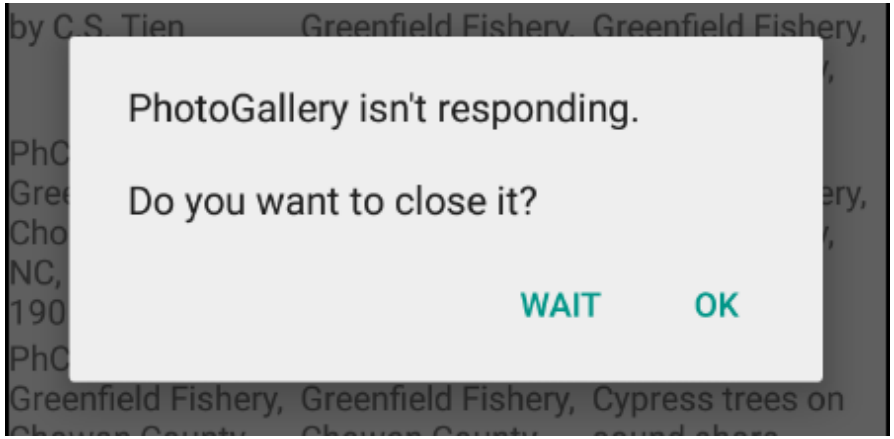
- Access Internet by using
  - URL and HttpURLConnection
  - WebView
  - Client-side APIs, such as the Google APIs, to interact directly with server processes.
- Download manager to download files
- XML parser to parse XML resources
  - Third-party tools to access Web Services
- Socket programming

Sample code

```
String myFeed = getString(R.string.my_feed);
try {
  URL url = new URL(myFeed);

  // Create a new HTTP URL connection
  URLConnection connection = url.openConnection();
  HttpURLConnection httpConnection = (HttpURLConnection)connection;

  int responseCode = httpConnection.getResponseCode();
  if (responseCode == HttpURLConnection.HTTP_OK) {
    InputStream in = httpConnection.getInputStream();
    processStream(in);
  }
}
catch (MalformedURLException e) {
  Log.d(TAG, "Malformed URL Exception.");
}
catch (IOException e) {
  Log.d(TAG, "IO Exception.");
}
```

Be responsive

- Android takes responsiveness very seriously.

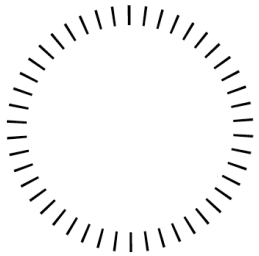


- General responsiveness rules:
  - An application must respond to any user action within 5 seconds.
  - A Broadcast Receiver must handle a broadcast message within 10 seconds.

Send email messages


```
//---sends an SMS message to another device---
private void sendEmail(String[] emailAddresses, String[] carbonCopies,
String subject, String message)
{
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:"));
    String[] to = emailAddresses;
    String[] cc = carbonCopies;
    emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
    emailIntent.putExtra(Intent.EXTRA_CC, cc);
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
    emailIntent.putExtra(Intent.EXTRA_TEXT, message);
    emailIntent.setType("message/rfc822");
    startActivity(Intent.createChooser(emailIntent, "Email"));
}
```

Quick exercise



- How to perform network operation asynchronously?
  - A. Perform socket communication directly in an activity.
  - B. Create a new thread to perform network operation.
  - C. Create a new service to perform network operation.
  - D. Create a new background task.

AsyncTask

- Define an asynchronous task

```
private class BackgroundTask extends AsyncTask
    <String, Void, Bitmap> {
        protected Bitmap doInBackground(String... url) {
         //---download an image---
        Bitmap bitmap = DownloadImage(url[0]);
        return bitmap;
    }

    protected void onPostExecute(Bitmap bitmap) {
        ImageView img = (ImageView) findViewById(R.id.img);
        img.setImageBitmap(bitmap);
    }
}
```

- Start an asynchronous task

```
new BackgroundTask().execute(
    "http://www.streetcar.org/mim/cable/images/cable-01.jpg");
```

Create a service

- Service runs in the background.

```
public class MyService extends Service {
    <service android:name=".MyService">
        <intent-filter>
            <action android:name="net.learn2develop.MyService" />
        </intent-filter>
    </service>
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        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();
    }
}
```

Start and stop a service


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

    Button btnStart = (Button) findViewById(R.id.btnStartService);
    btnStart.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startService(new Intent(getBaseContext(), MyService.class));
        }
    });

    Button btnStop = (Button) findViewById(R.id.btnStopService);
    btnStop.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            stopService(new Intent(getBaseContext(), MyService.class));
        }
    });
}
```

```
public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentServiceName");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Downloaded " + result + " bytes");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    private int DownloadFile(URL url) {
        try {
            //---simulate taking some time to download a file---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return 100;
    }
}
```

Quick exercise

- A. You must call stopSelf() or stopService() to stop a Intent Service once your job is done.
- B. An IntentService can execute multiple tasks concurrently.
- C. A bound service would run as long as the other application components are bound to it.
- D. An unbound service will stop if the the original component that initiated it is destroyed.

Perform repeated tasks

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // We want this service to continue running until it is explicitly
    // stopped, so return sticky.
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    doSomethingRepeatedly();
    return START_STICKY;
}

private void doSomethingRepeatedly() {
    timer.scheduleAtFixedRate( new TimerTask() {
        public void run() {
            Log.d("MyService", String.valueOf(++counter));
        }
    }, 0, UPDATE_INTERVAL);
}

@Override
public void onDestroy() {
    super.onDestroy();
    if (timer != null){
        timer.cancel();
    }
    Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
}
```

```
private Timer timer = new Timer();
```

Quick exercise

- If the time interval specified in scheduleAtFixedRate() is 1 second.
- If the code within the run() method takes 2 seconds to complete.
- How long will we wait until the run() method can be executed again?
  - A. 1 second
  - B. 2 seconds
  - C. 0 second
  - D. None of the above