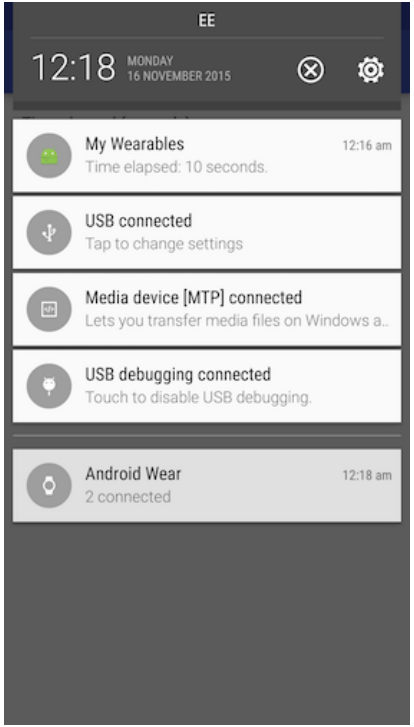
Services and Wearables

Lab 1

- Services
- Broadcasting
- Notifications



Services

A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface.

Three different types

- Scheduled

  API such as the JobScheduler launches the service

- Started

  can run in the background indefinitely

- Bound

  client-server interface that allows components to interact

Two super classes

`Service`

This is the base class for all services. When you extend this class, it's important to create a new thread in which the service can complete all of its work; the service uses your application's main thread by default, which can slow the performance of any activity that your application is running.

`IntentService`

This is a subclass of `Service` that uses a worker thread to handle all of the start requests, one at a time. This is the best option if you don't require that your service handle multiple requests simultaneously. Implement `onHandleIntent()`, which receives the intent for each start request so that you can complete the background work.

Extend IntentService

```
public class CountingService extends IntentService {

public static final String REPORT_KEY = "REPORT_KEY";
public static final String INTENT_KEY = "com.example.jianhuayang.mywearables.BROADCAST";

public CountingService() {
    super("BackgroundCounting");
}

@Override
protected void onHandleIntent(Intent intent) {

    int count = 0;
    while (count < 10) {
        synchronized (this) {
            try {
                wait(1000);
                count++;
                Log.d(MainActivity.DEBUG_KEY, Integer.toString(count));
            } catch (Exception e) {
            }
        }
    }
    Log.d(MainActivity.DEBUG_KEY, "service finished");

}
}
```

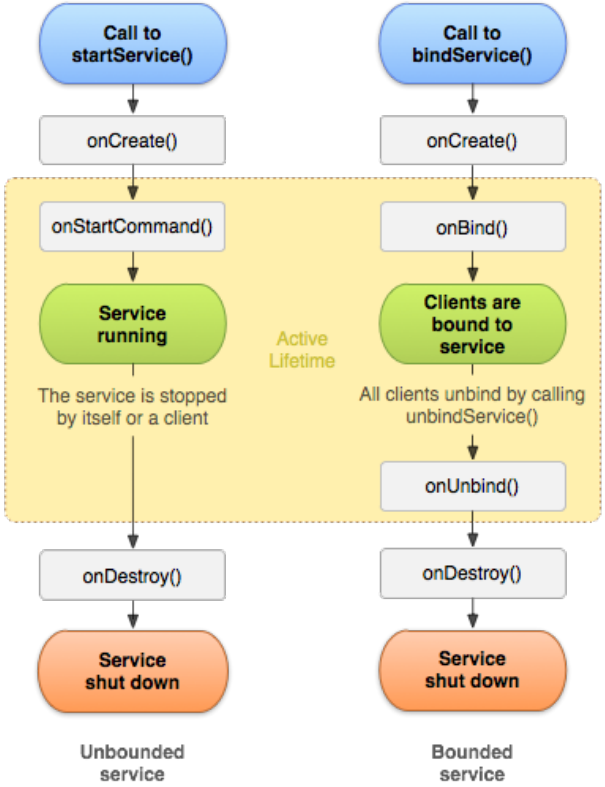
```
/**
 * The IntentService calls this method from the default worker thread with
 * the intent that started the service. When this method returns, IntentService
 * stops the service, as appropriate.
 */
```

In menifest

```
<service
        android:name=".CountingService"
        android:enabled="true"
        android:exported="false"
/>
```

To ensure that your app is secure, always use an explicit intent when starting a Service and do not declare intent filters for your services.

Service lifecycle



Broadcasting

```
Intent localIntent = new Intent();
localIntent.setAction(INTENT_KEY);
localIntent.putExtra(REPORT_KEY, Integer.toString(count));
sendBroadcast(localIntent);
Log.d(MainActivity.DEBUG_KEY, "broadcasted");
```

```
public class CountingReceiver extends BroadcastReceiver {
    public CountingReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(MainActivity.DEBUG_KEY, "on receive");
        String timeElapsed = intent.getStringExtra(CountingService.REPORT_KEY);
        Log.d(MainActivity.DEBUG_KEY, "time elapsed: " + timeElapsed);

        Intent intentNew = new Intent(context, DisplayActivity.class);
        intentNew.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intentNew.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        intentNew.putExtra(CountingService.REPORT_KEY, timeElapsed);
        context.startActivity(intentNew);
    }
}
```

Broadcasting

```
<receiver
        android:name=".CountingReceiver"
        android:enabled="true"
        android:exported="false">
        <intent-filter>
            <action android:name="com.example.jianhuayang.mywearables.BROADCAST" />
        </intent-filter>
</receiver>
```

LocalBroadcastManager

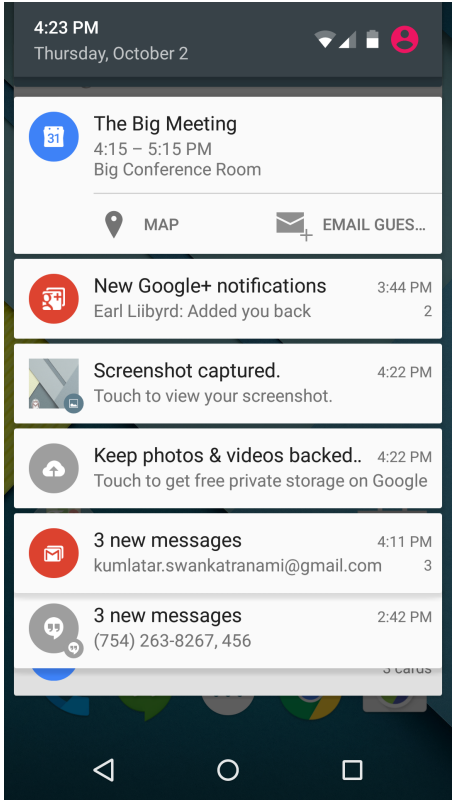
Notifications

- You specify the UI information and actions for a notification in a NotificationCompat.Builder object.
- To create the notification itself, you call NotificationCompat.Builder.build(), which returns a Notificationobject containing your specifications.
- To issue the notification, you pass the Notification object to the system by calling NotificationManager.notify().

Notifications

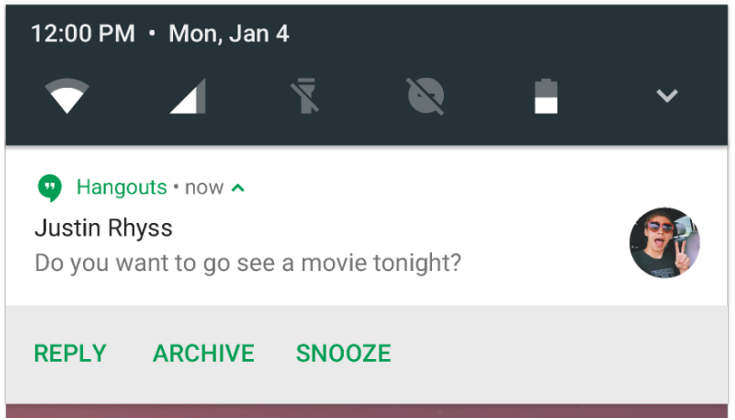
A `Notification` object *must* contain the following:

- A small icon, set by `setSmallIcon()`
- A title, set by `setContentTitle()`
- Detail text, set by `setContentText()`



Notification actions

- If you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling setContentIntent().
- You can also start an Activity when the user dismisses a notification.



PendingIntent

- By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity).
- A PendingIntent itself is simply a reference to a token maintained by the system describing the original data used to retrieve it

Preserving Navigation

- Regular activity
- Special activity

  The user only sees this Activity if it's started from a notification.

Regular activity

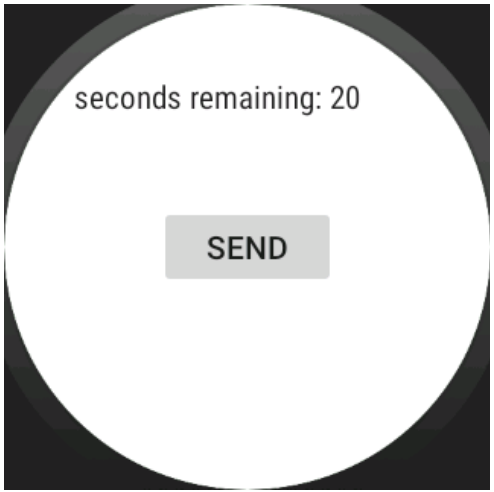
```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".ResultActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

Regular activity

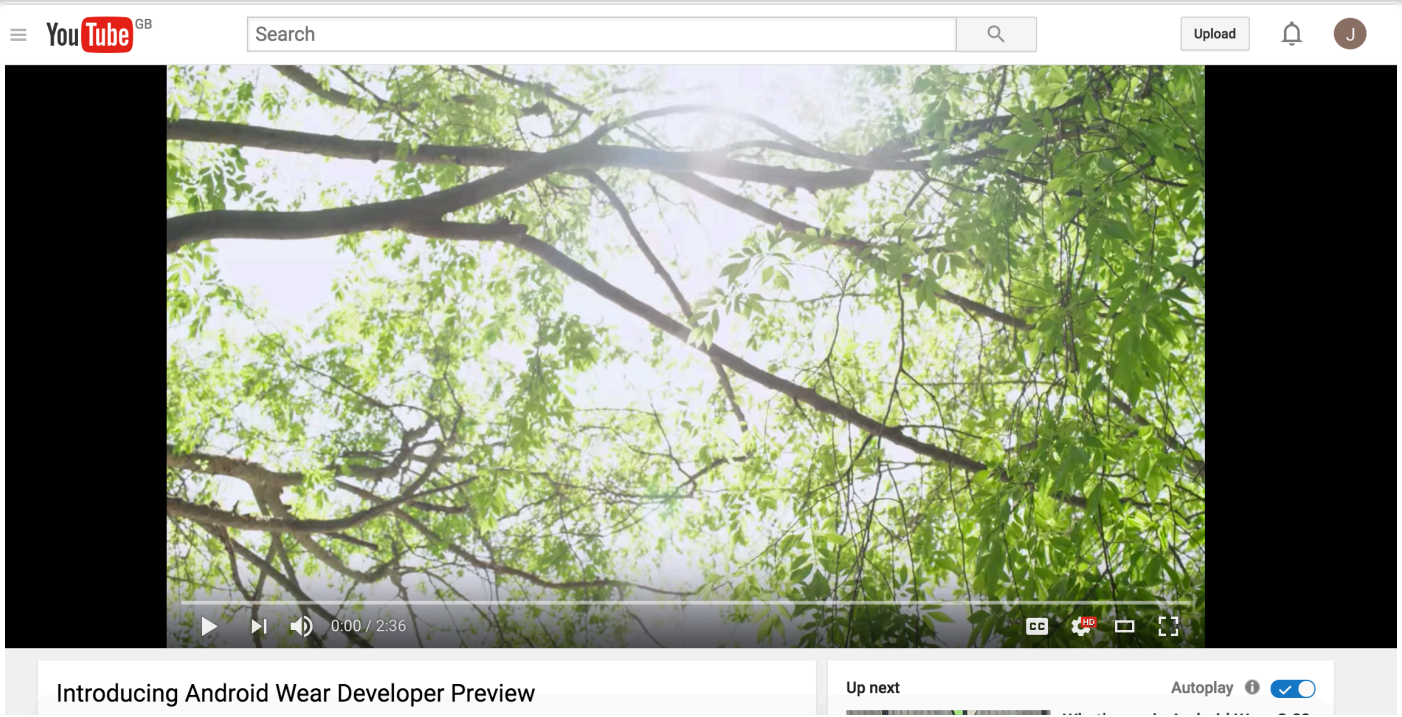
```
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

Lab 2

- Setting up
- Ambient mode
- Layout
- Send message

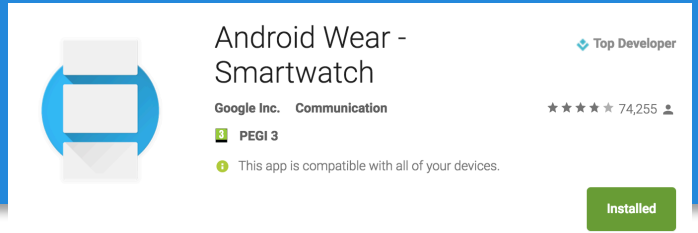


Introducing Android Wear



https://www.youtube.com/watch?v=0xQ3y902DEQ

Setting up



1. An Android phone/tablet, and
2. An Android Wear, either real or AVD
   - For AVD: adb -d forward tcp:5601 tcp:5601
   - For real:
     - Enable debug options
     - adb forward tcp:4444 localabstract:/adb-hub
     - adb connect 127.0.0.1:4444

Ambient mode

- Wear apps that run in both ambient and interactive mode are called *always-on* apps.

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

- If the user does not interact with your app for a period of time while it is displayed, or if the user covers the screen with their palm, the system switches the activity to ambient mode.

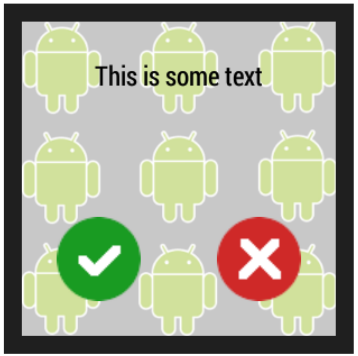
Ambient mode

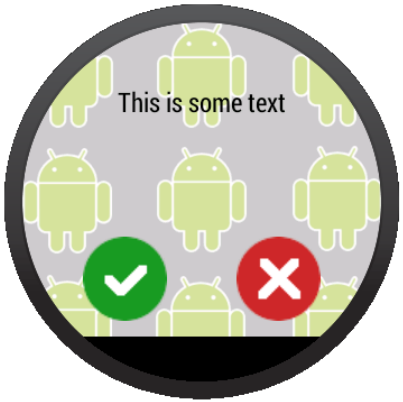
```
@Override
public void onEnterAmbient(Bundle ambientDetails) {
    super.onEnterAmbient(ambientDetails);
    updateDisplay();
}

@Override
public void onUpdateAmbient() {
    super.onUpdateAmbient();
    updateDisplay();
}

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

```
private void updateDisplay() {
    if (isAmbient()) {
        mContainerView.setBackgroundColor(getResources().getColor(android.R.color.black));
        mTextView.setTextColor(getResources().getColor(android.R.color.white));
        mClockView.setVisibility(View.VISIBLE);
        mClockView.setText(AMBIENT_DATE_FORMAT.format(new Date()));
        button.setTextColor(getResources().getColor(android.R.color.white));
    } else {
        mContainerView.setBackground(null);
        mTextView.setTextColor(getResources().getColor(android.R.color.black));
        mClockView.setVisibility(View.GONE);
        button.setTextColor(getResources().getColor(android.R.color.black));

    }
}
```

Shape-Aware Layout

BoxInsetLayout





LinearLayout

BoxInsetLayout

The BoxInsetLayout class included in the Wearable UI Library extends FrameLayout and lets you define a single layout that works for both square and round screens.

```
<android.support.wearable.view.BoxInsetLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@drawable/robot_background"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:padding="15dp">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="5dp"
        app:layout_box="all">

        <TextView
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="@string/sometext"
            android:textColor="@color/black" />

        <ImageButton
            android:background="@null"
            android:layout_gravity="bottom|left"
            android:layout_height="50dp"
            android:layout_width="50dp"
            android:src="@drawable/ok" />

        <ImageButton
            android:background="@null"
            android:layout_gravity="bottom|right"
            android:layout_height="50dp"
            android:layout_width="50dp"
            android:src="@drawable/cancel" />
    </FrameLayout>
</android.support.wearable.view.BoxInsetLayout>
```

Send data

- Wearable Data Layer API, which is part of Google Play services, provides a communication channel for your handheld and wearable apps
- Many choices for different purposes

MessageApi & WearableListenerService

- The MessageApi class can send messages and is good for remote procedure calls (RPC), such as controlling a handheld's media player from the wearable or starting an intent on the wearable from the handheld.
- Extending WearableListenerService lets you listen for important data layer events in a service. The system manages the lifecycle of the WearableListenerService, binding to the service when it needs to send data items or messages and unbinding the service when no work is needed.

Send message

```
private void initApi() {
    client = getGoogleApiClient(this);
    retrieveDeviceNode();
}

private GoogleApiClient getGoogleApiClient(Context context) {
    return new GoogleApiClient.Builder(context)
            .addApi(Wearable.API)
            .build();
}

private void retrieveDeviceNode() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            client.blockingConnect(CONNECTION_TIME_OUT_MS, TimeUnit.MILLISECONDS);
            NodeApi.GetConnectedNodesResult result =
                    Wearable.NodeApi.getConnectedNodes(client).await();
            List<Node> nodes = result.getNodes();
            if (nodes.size() > 0) {
                nodeId = nodes.get(0).getId();
                Log.d("DEBUG_KEY", "size "+Integer.toString(nodes.size()));

                Log.d("DEBUG_KEY", nodeId);
            }
            client.disconnect();
        }
    }).start();
}
```

```
public void onSendClick(View v) {
    if (nodeId != null) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                Log.d("DEBUG_KEY", "on click");
                client.blockingConnect(CONNECTION_TIME_OUT_MS, TimeUnit.MILLISECONDS);
                Wearable.MessageApi.sendMessage(client, nodeId, message, null);
                client.disconnect();
                Log.d("DEBUG_KEY", "on click sent");

            }
        }).start();
    }
}
```

Listen to message

```
public class ListenerService extends WearableListenerService {
    @Override
    public void onMessageReceived(MessageEvent messageEvent) {
        Log.d("DEBUG_KEY", "on received");
        showToast(messageEvent.getPath());
    }

    private void showToast(String message) {
        Toast.makeText(this, message, Toast.LENGTH_LONG).show();
    }
}
```

```
<service
        android:name=".ListenerService">
        <intent-filter>
            <action android:name="com.google.android.gms.wearable.BIND_LISTENER" />
        </intent-filter>
</service>
```

Debug wear apps

- Production:
  - when users download the handheld app, the system automatically pushes the wearable app to the paired wearable.
- Debugging:
  - While developing, installing apps with adb install or Android Studio directly to the wearable is required.

References

- Android API guide on Services
- Android Best Practices for Background Jobs
- Building Apps for Wearables
- Notifications https://developer.android.com/guide/topics/ui/notifiers/notifications.html