SENG440
Week 10

Connectivity, multitouch, and gestures

Interacting with other phones

- **Connectivity** means we can design apps that **enable interaction between two or more phones**
  - In the same vicinity.
  - Over the internet– network connection to shared web server.
- **Google Nearby API** simplifies nearby communication via multiple methods -- Nearby | Google for Developers
  - Bluetooth - ~2.4 GHz frequency range, ~10-20m indoors
  - WiFi aware – cluster of neighboring devices – faster throughput than Bluetooth
  - Near field communication (NFC) – 4cm distance
  - Audio - ultrasound
- ~100m radio range (including wifi)

Nearby

- Two main APIs
  - **Connectivity** – peer-to-peer networking API
  - **Messaging** – publish and subscribe to small payloads of data
- Not encrypted
- Two phases - pre-connection and post-connection
  - **Advertisers** – advertise themselves
  - **Discoverers** – discover other devices that are advertising
  - **No distinction** between advertisers and discoverers **once a connection** has been established. (Both can be servers and clients).

Connection strategies

- **P2P_CLUSTER**
  - M-to-N, or cluster-shaped, connection topology
  - Amorphous cluster
  - Lower bandwidth (small packets of information)
- **P2P_STAR**
  - 1-to-N, or star-shaped, connection topology
  - Each device is either a hub or spoke but not both
  - Higher bandwidth (e.g. sharing videos)
- **P2P_POINT_TO_POINT**
  - 1-to-1 connection topology
  - Highest bandwidth

Advertise and discover

- On advertiser call `startAdvertising()` with the desired `Strategy` and a `serviceId`
- On discovering device call `startDiscovery()` with the same `Strategy` and `serviceId`
- Best to use package name for `serviceId`
- Implement a `ConnectionLifecycleCallback`, called when connection happens

```java
private void startAdvertising() {
  AdvertisingOptions advertisingOptions =
      new AdvertisingOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startAdvertising(
          getUserNickname(), SERVICE_ID, connectionLifecycleCallback, advertisingOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're advertising!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We were unable to start advertising.
          });
}
```

Advertise and discover

- Discoverer implements an `EndpointDiscoveryCallback`

```
private void startDiscovery() {
  DiscoveryOptions discoveryOptions =
      new DiscoveryOptions.Builder().setStrategy(STRATEGY).build();
  Nearby.getConnectionsClient(context)
      .startDiscovery(SERVICE_ID, endpointDiscoveryCallback, discoveryOptions)
      .addOnSuccessListener(
          (Void unused) -> {
            // We're discovering!
          })
      .addOnFailureListener(
          (Exception e) -> {
            // We're unable to start discovering.
          });
}
```

Stopping advertising and discovery

- `stopAdvertising()` when you no longer need to advertise
- `stopDiscovery()` when you no longer need to discover
- However:
  - Advertiser can still accept connections from discoverers that found it before it stopped, and
  - Discoverer can still request connections to advertisers already found, but will not discover any more newly advertised
  - Call `stopDiscovery` after connection is made because discovery operation uses radio communication that can disrupt communication

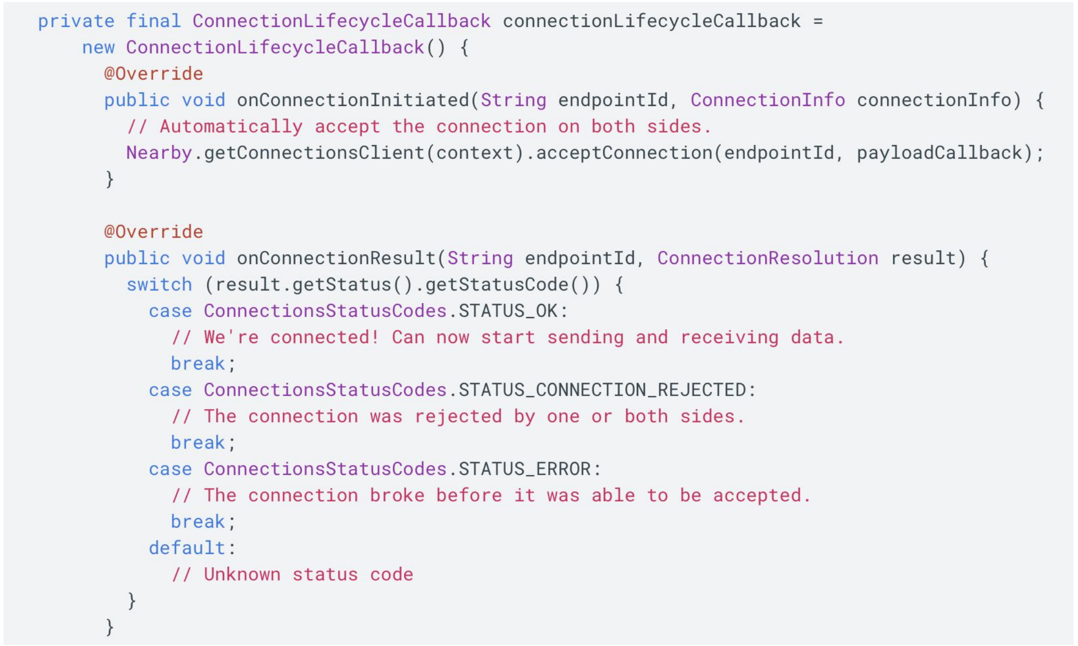
Making connections

- Advertise and discover only make devices available to each other, but do not initiate connection
- Callback function can initiate connection

```java
private final EndpointDiscoveryCallback endpointDiscoveryCallback =
    new EndpointDiscoveryCallback() {
      @Override
      public void onEndpointFound(String endpointId, DiscoveredEndpointInfo info) {
        // An endpoint was found. We request a connection to it.
        Nearby.getConnectionsClient(context)
            .requestConnection(getUserNickname(), endpointId, connectionLifecycleCallback)
            .addOnSuccessListener(
                (Void unused) -> {
                  // We successfully requested a connection. Now both sides
                  // must accept before the connection is established.
                })
            .addOnFailureListener(
                (Exception e) -> {
                  // Nearby Connections failed to request the connection.
                });
      }

      @Override
      public void onEndpointLost(String endpointId) {
        // A previously discovered endpoint has gone away.
      }
    };
```

Accepting / rejecting connections

- After connection initiated `onConnectionInitiated()` method of the `ConnectionLifecycleCallback` callback is called
- Both sides must choose to accept or reject the connection
- Can also add authentication tokens (recommended)



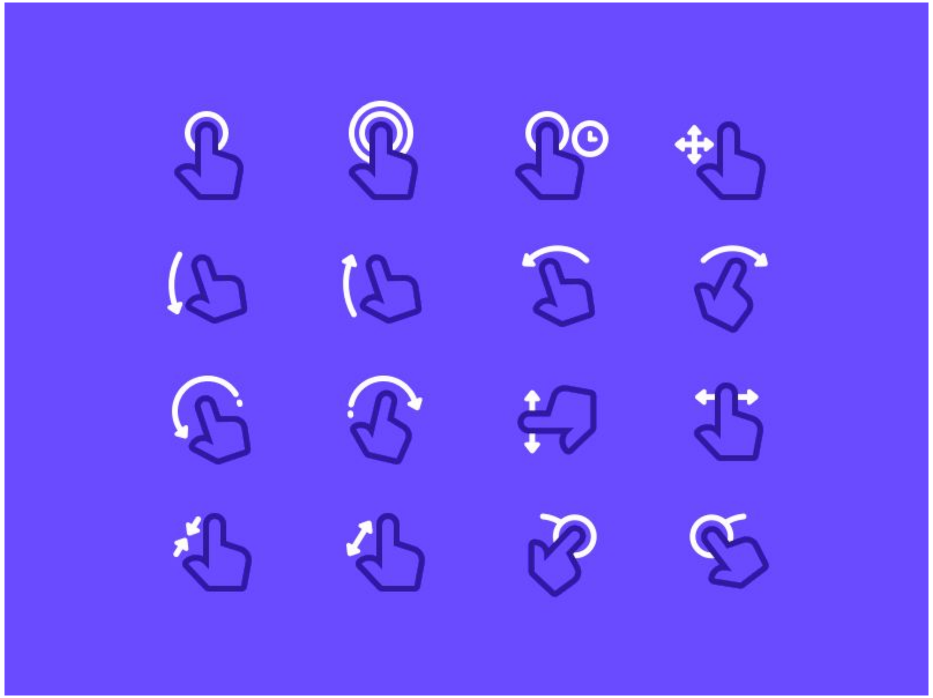
```java
private final ConnectionLifecycleCallback connectionLifecycleCallback =
    new ConnectionLifecycleCallback() {
      @Override
      public void onConnectionInitiated(String endpointId, ConnectionInfo connectionInfo) {
        // Automatically accept the connection on both sides.
        Nearby.getConnectionsClient(context).acceptConnection(endpointId, payloadCallback);
      }

      @Override
      public void onConnectionResult(String endpointId, ConnectionResolution result) {
        switch (result.getStatus().getStatusCode()) {
          case ConnectionsStatusCodes.STATUS_OK:
            // We're connected! Can now start sending and receiving data.
            break;
          case ConnectionsStatusCodes.STATUS_CONNECTION_REJECTED:
            // The connection was rejected by one or both sides.
            break;
          case ConnectionsStatusCodes.STATUS_ERROR:
            // The connection broke before it was able to be accepted.
            break;
          default:
            // Unknown status code
        }
      }
```

Sending payloads

- Bytes, Files, Streams
- sent using the `sendPayload()` method
- received in an implementation of `PayloadCallback` that's passed to `acceptConnection()`
- See: Exchange data | Nearby Connections | Google for Developers

Motion and Gestures

- Touch based `MotionEvents` are sent by Android system
- Implement a new `View` that has motion and gesture listeners



Motion events

- We use `MotionEvents` to handle touch events on the screen
  - `MotionEvent.ACTION_DOWN`
    - A pressed gesture has started, the motion contains the initial starting location
  - `MotionEvent.ACTION_POINTER_DOWN`
    - A non-primary pointer has gone down.
  - Others, see: MotionEvent | Android Developers
- Override `onTouchEvent(event: MotionEvent)` in View

Gesture detector

- `GestureDetector` detects common gestures, built on top of motion events:
  - onDown
  - onFling
  - onLongPress
  - onScroll
  - onSingleTapUp
- `GestureDetector.SimpleOnGestureListener` overrides all `on<TouchEvent>` methods. You override only the ones you need in your gesture listener sub-class.
- When you override `onTouchEvent` pass the event to the listener
- `ScaleGestureDetector` convenience class to listen for a subset of scaling-related events

Common uses of gestures

- Gesture experience should be **similar across different apps**
- **Navigational** gestures
  - Tap, scroll and pan, drag, swipe, pinch
- **Action** gestures
  - Tap, long press, swipe
- **Transform** gestures
  - Double tap, pinch, compound gestures, pick up and move (long press + drag)
- **Selection**
  - Long press touch, two-finger touch, shortcut such as tapping an icon
- **Swipe-to-refresh**

https://developer.android.com/develop/ui/views/touch-and-input/swipe

Extra: Creating preference screens (settings)

- Preferences is now part of the AndroidX library, add this to your build.gradle file:
  - androidx.preference:preference:1.2.0
- Create `xml/preferences.xml` resource file.
- Subclass `PreferenceFragmentCompat` and override `onCreatePreferences` method
- Reference the preferences fragment in your Activity layout xml
- Access using the `prefs` property in your `Activity`
- Still need to make your own preference screen for Jetpack Compose

TODO for next week

- Read the Android guide on how to Use touch gestures, and take a look at the examples on the Gestures design guide.
- Read the article Gestural Interfaces: A Step Backward In Usability from 2010 and think about questions/comments to discuss next week.
- Some ideas to think about:
  - Do you think that the problems presented in that article have been solved in the last 10 years when you look at the latest design guide?
  - How might these issues manifest differently in different kinds of mobile devices: phones, smart watches, etc.?
  - Can you imagine new ways to use sensors or the camera for user input?