Lab on apps development for tablets, smartphones and smartwatches

Week 3: Activities and Intents

Dr. Marina Zapater, Prof. David Atienza

Ms. Elisabetta de Giovanni, Ms. Halima Najibi, Ms. Farnaz Forooghifar

Mr. Grégoire Surrel, Mr. Dionisijie Sopic

Embedded Systems Laboratory (ESL) – Faculty of Engineering (STI)

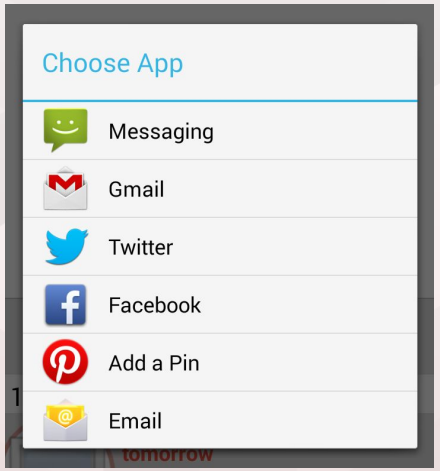
Today: catch-up week!

- All labs we'll build on top of the SportsTracker app
  - At the end of this lab, we need to have a good skeleton for the app
  - … and understand Activities and Intents.
- Today's lab
  - Activities and intents
  - Communication with Android Wear
- Very short lecture today!
  - Only "intents" → needed for our base app

Class outline

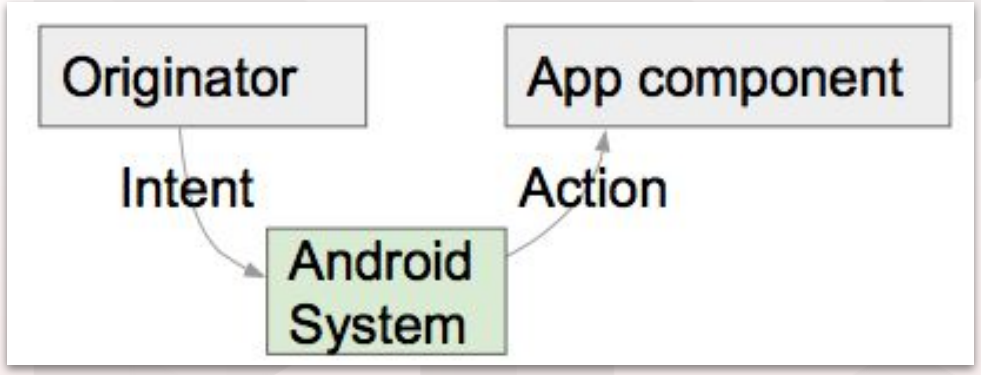
- **Intents:**
  - Explicit intents: how to launch activities
  - Implicit intents
  - Sending data to indents
  - Retrieving results

- Communication with Android Wear



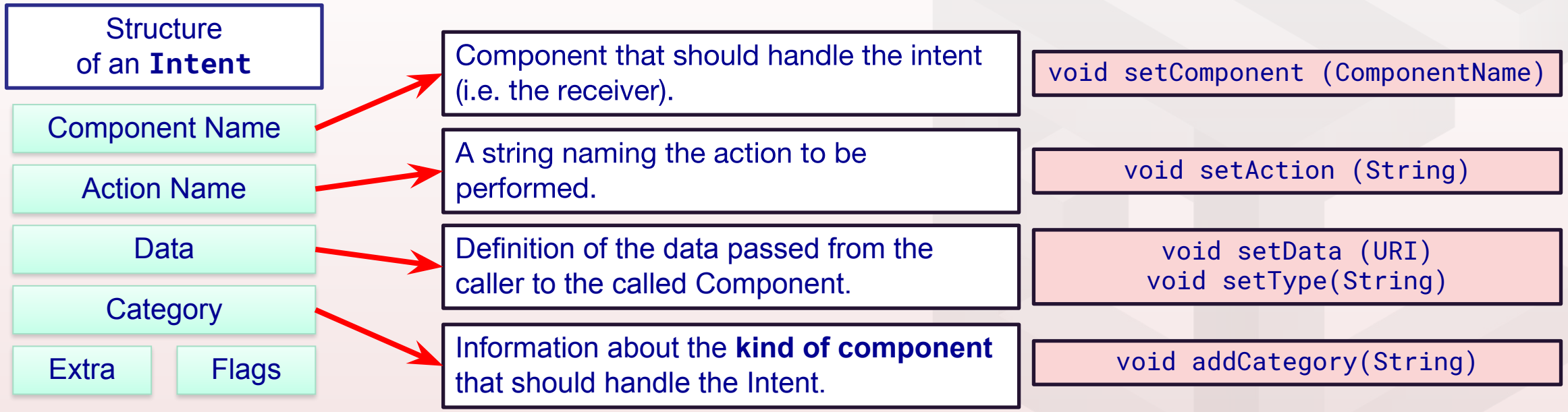
What is an Intent?

- An intent is a description of an operation to be performed
- Is an object used to request an action from another app component via the Android system
  - And to pass data between components
- Used to:
  - Start activities:
    - A button click starts a new activity
    - Clicking “Camera” opens the camera app
  - Start services:
    - Initiate downloading a file in the background
  - Deliver broadcast messages:
    - The system informs the phone is charging



Structure and definition of an Intent

- We can think of an **Intent** object as a **message** containing a bundle of information.
  - Information of interest for the receiver (e.g. data)
  - Information of interest for the Android system (e.g. category).



Structure and definition of an Intent

- We can think of an **Intent** object as a **message** containing a bundle of information.
  - Information of interest for the receiver (e.g. data)
  - Information of interest for the Android system (e.g. category).

Structure of an **`Intent`**

- Component Name
- Action Name
- Data
- Category
- Extra
- Flags

Additional information that should be delivered to the handler (e.g. parameters).

Key-value pairs

```
void putExtras()
getExtras()
```

Explicit and implicit intents

- Implicit Intent:
  - Asks system to find an activity that can handle this request
    - Clicking “Share” opens a chooser with a list of apps.
  - The received is specified by “data type/names”

- Explicit intent:
  - Starts a specific activity
  - The receiver is specified through the “Component name”
  - **Used to launch activities!**

Sending an Implicit intent

- Showing a web page:

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

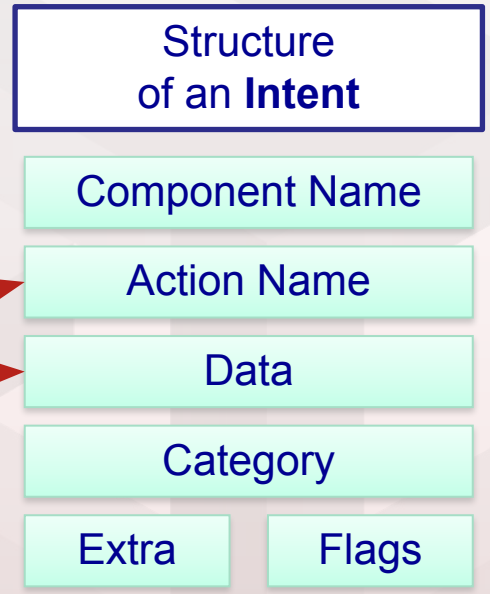
- Dial a phone number:

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

- The picture chooser in Lab3:

```
private static final int PICK_IMAGE = 1;

public void chooseImage(View view) {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(
            Intent.createChooser(intent, title: "Select Picture"), PICK_IMAGE);
}
```



Starting Activities with Explicit Intents

To start an activity, we use an explicit intent:

1. Create an intent:
   - `Intent intent = new Intent (this, ActivityName.class);`
2. Use the intent to start the activity
   - `startActivity(intent);`

```java
public class MainActivity extends AppCompatActivity {

    EditText e1;
    TextView t2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        e1 = (EditText) findViewById(R.id.edit1);
        t2 = (TextView) findViewById(R.id.t2);
    }


    public void clickButton2(View view) {
        t2.setText("I clicked button 2");
    }

    public void clickButton1(View view) {
        t2.setText("I clicked button 1");
    }

    public void doSomething(View view) {
        Intent i1 = new Intent(this, Second.class);
        i1.putExtra("user", e1.getText().toString());
        startActivity(i1);
    }
}
```

```java
public class Second extends AppCompatActivity {

    TextView txt1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        txt1 = (TextView) findViewById(R.id.result);

        Bundle b1 = getIntent().getExtras();
        String s1 = b1.getString("user");
        txt1.setText(s1);
    }
}
```

Sending and receiving data with Intents

- Two types of sending data with Intents:
  - Data: one piece of information whose data location can be represented by an URI
  - Extras: one or more pieces of information as a collection of key-value pairs in a Bundle

- Sending activity:
  1. Create Intent
  2. Put data or extras on intent
  3. Start Activity
- Receiving activity:
  1. Get the intent object
  2. Retrieve the data

Structure of an **Intent**

- Component Name
- Action Name
- Data
- Category
- Extra
- Flags

Sender

```java
public class MainActivity extends AppCompatActivity {

    public void doSomething(View view) {
        Intent i1 = new Intent(this, Second.class);
        i1.putExtra("user", e1.getText().toString());
        startActivity(i1);
    }
}
```

Receiver

```java
public class Second extends AppCompatActivity {

    TextView txt1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        txt1 = (TextView) findViewById(R.id.result);

        Bundle b1 = getIntent().getExtras();
        String s1 = b1.getString("user");
        txt1.setText(s1);
    }
}
```

Intent with Results

- Activities could return results → Useful to have some data back!
- **Sender side:**
  1. Invoke the **`startActivityForResult`**`(Intent intent, int requestCode)`
  2. Implement **`onActivityResult`**`(int requestCode, int resultCode, Intent data)`
     - Receives result via "`Intent receivedIntent`" using the Extras

1.

```java
// Usually at the top of the class
private static final int PICK_IMAGE = 1;

// Anywhere with the other methods of the class
public void chooseImage(View view) {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(intent, "Select Picture"),
            PICK_IMAGE);
}
```

2.

```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
                                                                 receivedIntent) {

    super.onActivityResult(requestCode, resultCode, receivedIntent);

    switch (requestCode) {
        case PICK_IMAGE:
            if (resultCode == RESULT_OK) {
                Intent intent = new Intent(this, ValidationActivity.class);
                intent.putExtra("imageUri", receivedIntent.getData());
                startActivityForResult(intent, VALIDATE_IMAGE);
                imageUri = receivedIntent.getData();
            }
            break;
    }
}
```

Intent with Results

- Activities could return results → Useful to have some data back!
- **Receiver side:**
  - In our example (Lab3), the activity launched by the implicit intent takes care of the putting the result available on the "`Intent receivedIntent`"
  - In case of an explicit intent, we need to invoke on the receiver `setResult()`:

```
void setResult(int resultCode, Intent data);
finish();
```

- This is what we do in the "Registering new user" part in Lab3:

Sender: `LoginActivity`
(calls `EditProfileActivity`)

```
Intent intent = new Intent( packageContext: LoginActivity.this,
        EditProfileActivity.class);
startActivityForResult(intent, REGISTER_PROFILE);
```

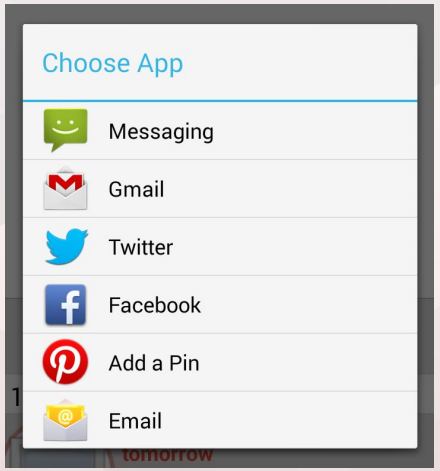
Receiver: `EditProfileActivity`

```
Intent intent = new Intent( packageContext: EditProfileActivity.this,
        LoginActivity.class);
intent.putExtra( name: "userProfile", userProfile);
setResult(AppCompatActivity.RESULT_OK, intent);
finish();
```

Class outline

- Intents:
  - Explicit intents: how to launch activities
  - Implicit intents
  - Sending data to indents
  - Retrieving results

- **Communication with Android Wear**



The Android Wear APIs

- Multiple communication channels:
  - The *Message* API
    - From **one node to another**
    - Arbitrary payload (as bytes)
    - Messages are sent to a specific path
    - Good for one-way requests
  - The *Data* API
    - From one node to **all connected nodes** (including self), **synchronized**
    - Arbitrary payload
    - Data is sent to a specific path
    - Good for structured data
  - The *Channel* API
    - Dedicated to big payloads (images, music)
    - No automatic synchronization
    - Suitable for **streams**

Will be used in today's lab!

The WearService.java class

- Uses both *Data* and *Message* API
- Data sent using intents

Sender: SomeActivity

```
/* Sending an intent to the WearService */
Intent intent = new Intent(this, WearService.class);
intent.setAction(WearService.ACTION_SEND.ACTION_NAME.name());
intent.putExtra(… /* Extras depends on the action to do */);
startService(intent);
```

Receiver: WearService

```
public int onStartCommand(Intent intent, int flags, int startId)
```

Message API

Data API

The `WearService.java` class

- Data received *also* using intents

Declaration in the `AndroidManifest.xml`

```xml
<service android:name=".WearService">
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
        <data android:host="*" android:pathPrefix="" android:scheme="wear" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.DATA_CHANGED" />
        <data android:host="*" android:pathPrefix="" android:scheme="wear" />
    </intent-filter>
</service>
```

Implementation in the `WearService`

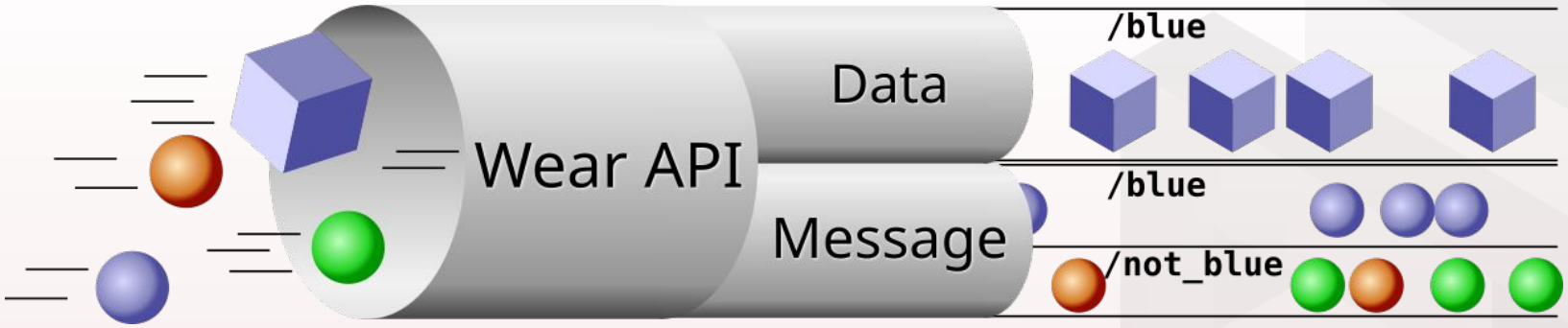

```java
@Override
public void onMessageReceived(MessageEvent messageEvent) {
    Log.v(TAG, msg: "onMessageReceived: " + messageEvent);
```

```java
@Override
public void onDataChanged(DataEventBuffer dataEvents) {
    Log.v(TAG, msg: "onDataChanged: " + dataEvents);
```

- Result/action using intents:
  - *Explicit* intent, such as “Start this `Activity`”
  - *Implicit* intent, local broadcast as “This image `Asset` was just received”

Wear API: paths and keys

- Data is sent, with a visual example (without the Channels API)
  - to a specific **path** (/blue or /not_blue)
  - with a given **key** (box or sphere, but could be something else)



- Real-world example: Sensors!
  - Send messages to path `/configure`, with keys *'sensor'*, *'screen'*, … and values *'enable'* and *'disable'*
  - Send messages to path `/measure`, with keys *'light'*, *'gyroscope'*, … and values being `float` values

Data flow using the WearService

Mobile app

1) Send an `Intent` to the `WearService`

```
Intent intent = new Intent( packageContext: this, WearService.class);
intent.setAction(WearService.ACTION_SEND.EXAMPLE_ASSET.name());
intent.putExtra(WearService.IMAGE, asset);
startService(intent);
```

2) In the `WearService`, get the `Intent` data and transmit it

```
case EXAMPLE_ASSET:
    putDataMapRequest = PutDataMapRequest.create(BuildConfig.W_example_path_asset);
    putDataMapRequest.getDataMap().putAsset(
            BuildConfig.W_image, (Asset) intent.getParcelableExtra(IMAGE));
    sendPutDataMapRequest(putDataMapRequest);
    break;
```

Wear app

1) Register for local broadcasts in the `Activity`

```
LocalBroadcastManager.getInstance(this).registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.v( tag: "MainActivity",  msg: "Received intent");
        ImageView imageView = findViewById(R.id.imageView);
        byte[] byteArray = intent.getByteArrayExtra(IMAGE_DATA_BYTES);
        Bitmap bmp = BitmapFactory.decodeByteArray(byteArray,  offset: 0, byteArray.length);
        imageView.setImageBitmap(bmp);
    }
}, new IntentFilter(IMAGE_BROADCAST));
```

2) In the `WearService`, receive the data from the `onDataChanged()` *[or `onMessageReceived()` if it's a message]*

```
case BuildConfig.W_example_path_asset:
    // Extract the data behind the key you know contains data
    Asset asset = dataMapItem.getDataMap().getAsset(BuildConfig.W_image);
    intent = new Intent(MainActivity.IMAGE_BROADCAST);
    bitmapFromAsset(asset, intent, MainActivity.IMAGE_DATA_BYTES);
    break;
```

3) The `LocalBroadcastManager` (hidden in the `bitmapFromAsset()` call in this example) will trigger the `BroadcastReceiver` registered in step 1

```
LocalBroadcastManager.getInstance(WearService.this).sendBroadcast(intent);
```

Questions?