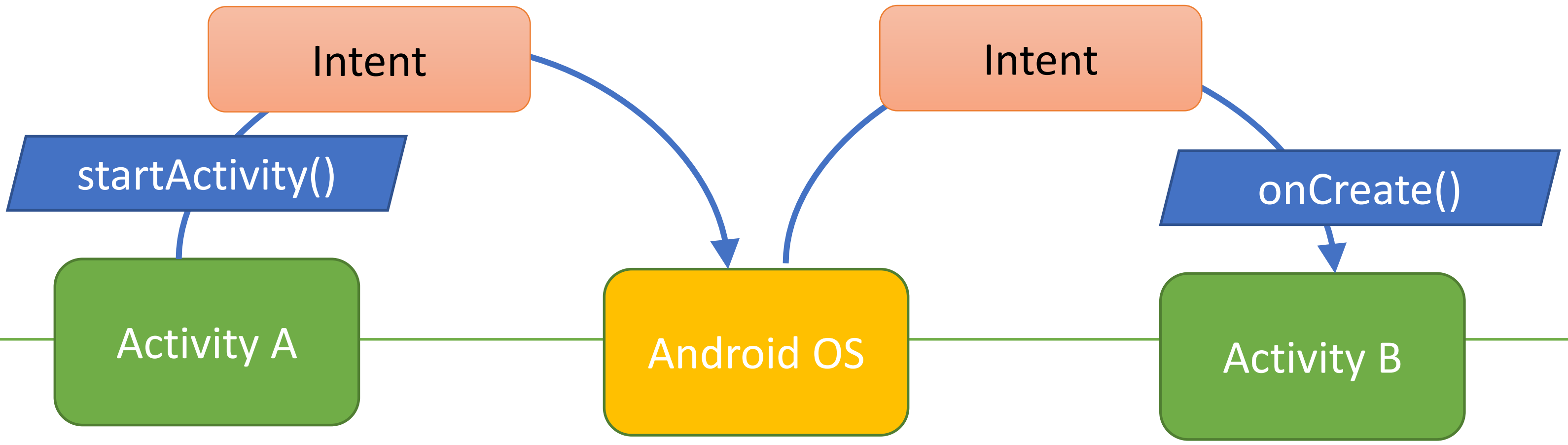
Intents

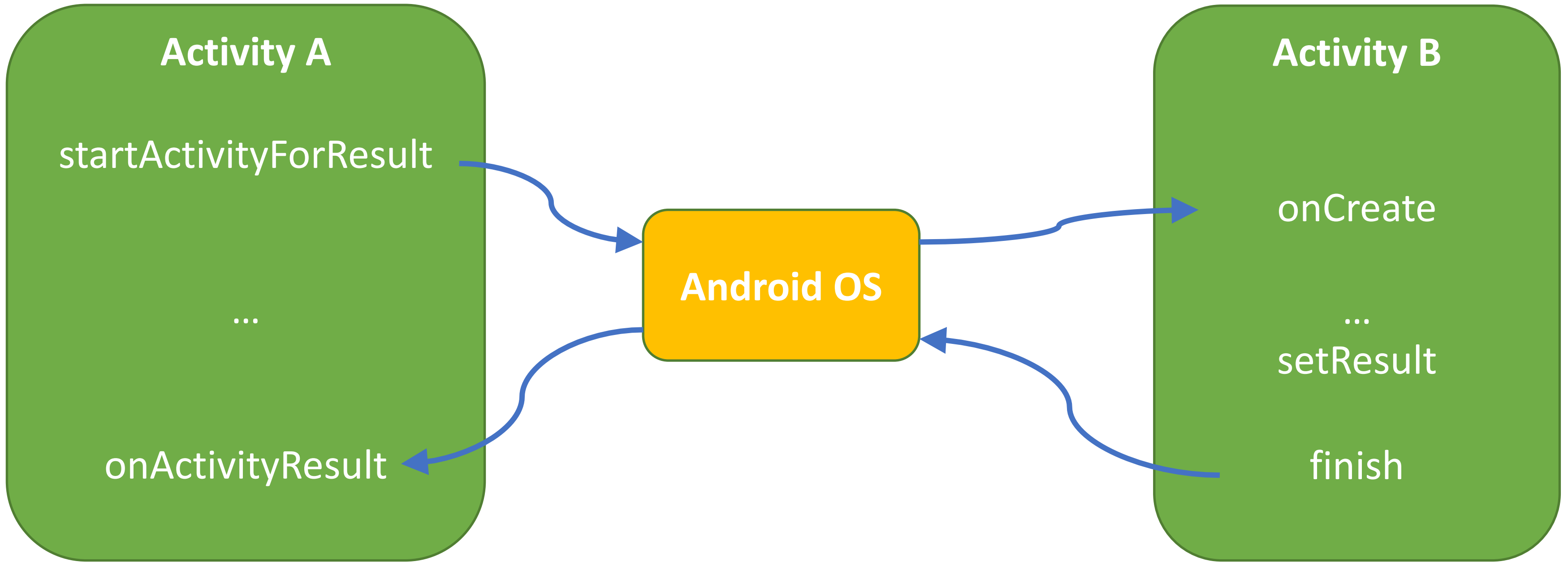
Intents

- **Intent**: a message used by a component to request action from another app or component
- 3 main use cases for Intents
- **Case 1** (Activity A starts Activity B, no result back):
  - Call *startActivity()*, pass an Intent
  - Intent has information about Activity to start, plus any necessary data



Intents

- **Case 2** (Activity A starts Activity B, gets result back):
  - Call *startActivityForResult()*, pass an Intent
  - Separate Intent received in Activity A's *onActivityResult()* callback



Intents

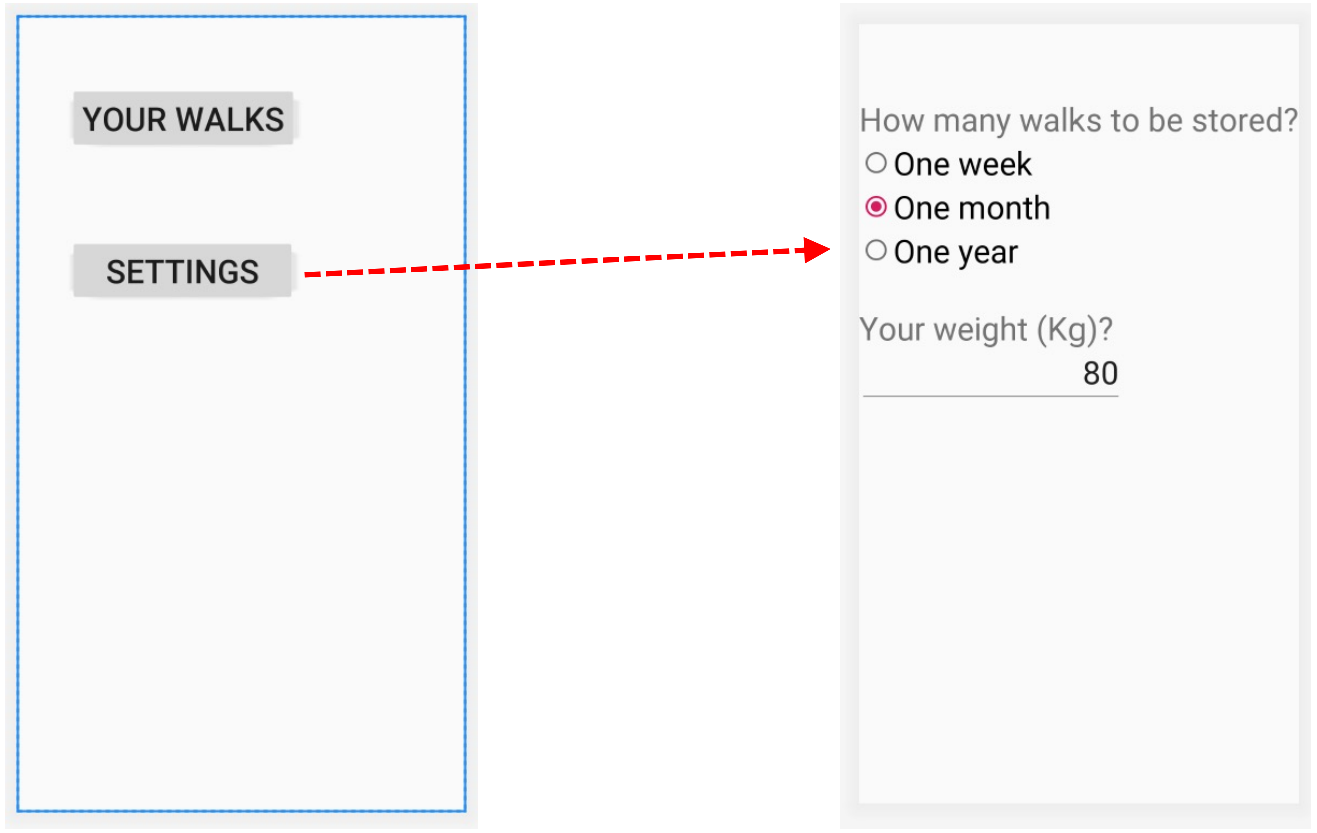
- **Case 3** (Activity A starts a Service):
  - E.g. Activity A starts service to download big file in the background
  - Activity A calls *startService()*, passes an Intent
  - Intent contains information about Service to start, plus any necessary data

Implicit/explicit intents

- **Explicit Intent**: If components sending and receiving Intent are in same app
  - E.g. Activity A starts Activity B in same app
  - Activity A explicitly says what Activity (B) should be started
- **Implicit Intent**: If components sending and receiving Intent are in different apps
  - Activity A specifies what action it needs to be done, doesn’t specify Activity to do it
  - Example of action: take a picture, any camera app can be ok

Intents: example

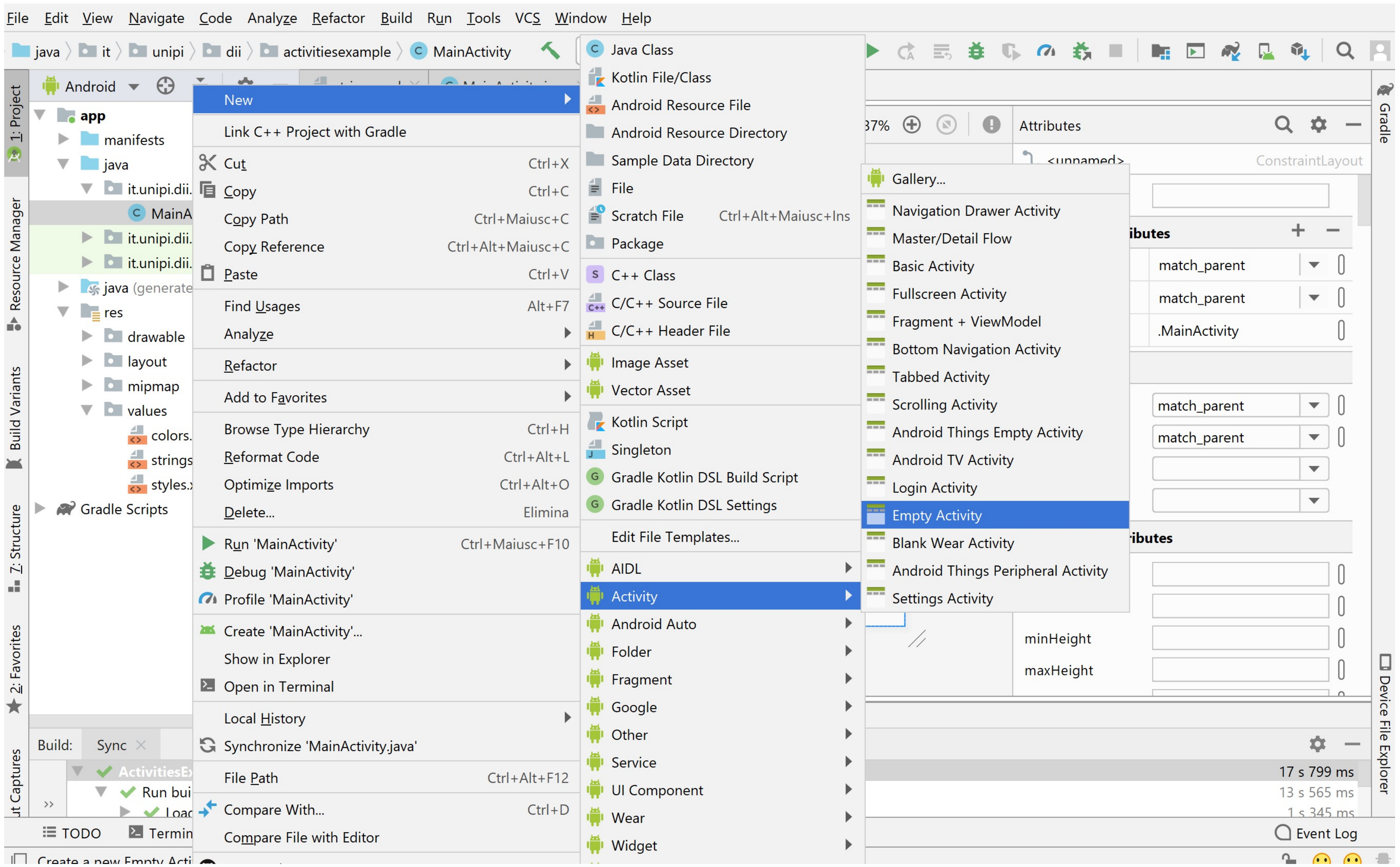
- Goal: start a second activity when a button is pressed
- Let’s suppose the first activity is part of an app about monitoring user’s walks



How many walks to be stored?
○ One week
[✓ One month]
○ One year

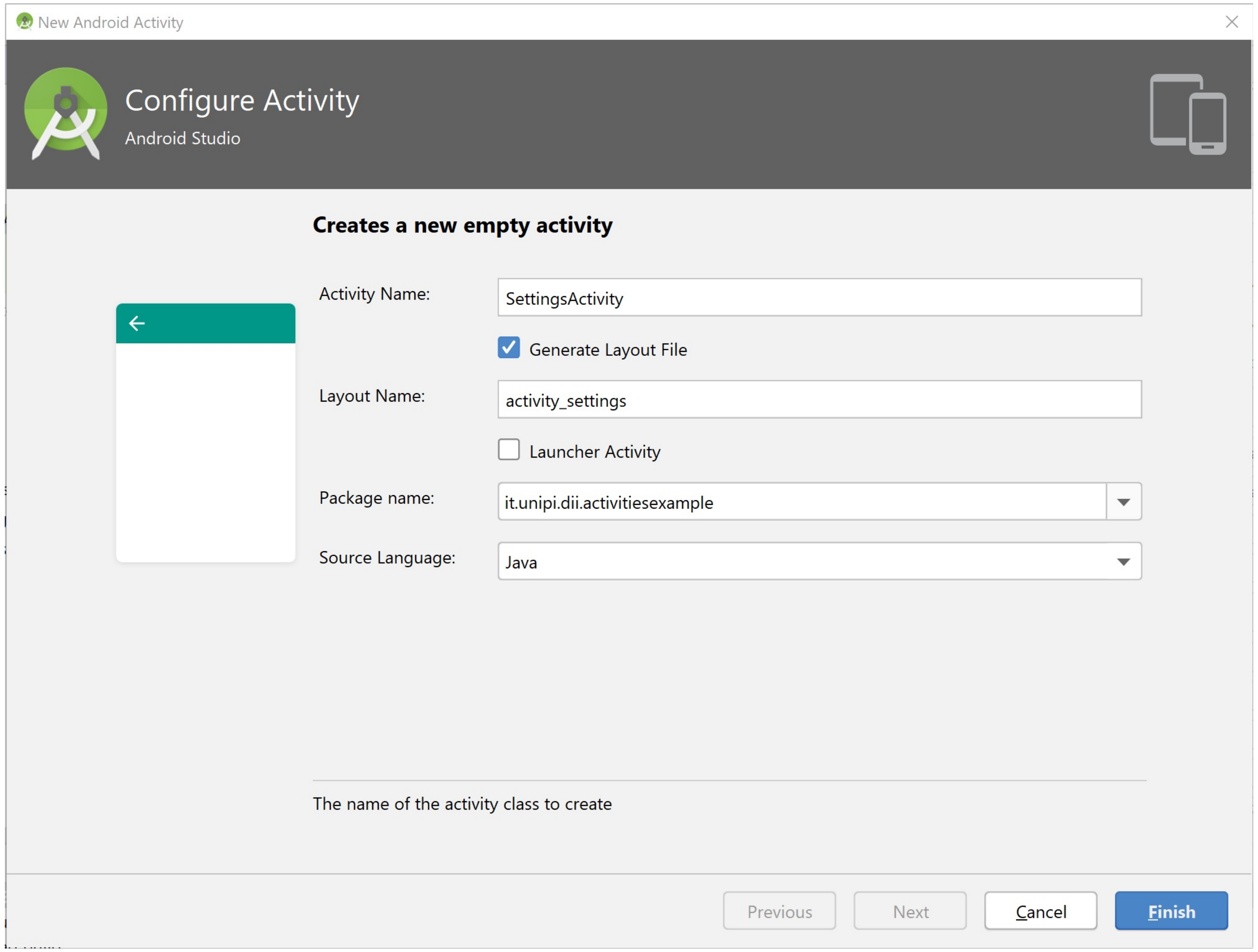
Your weight (Kg)?
80

Create new activity in Android Studio



Configure new Activity

- Provide name and layout file for the new activity


New Android Activity

Configure Activity

Android Studio

Creates a new empty activity

Activity Name: SettingsActivity

[✓] Generate Layout File

Layout Name: activity_settings

[ ] Launcher Activity

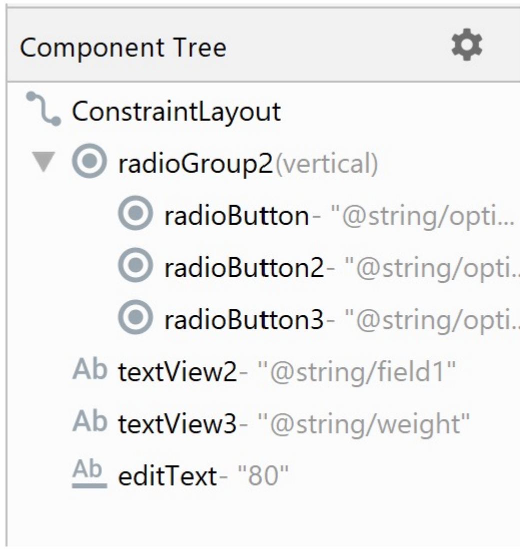
Package name: it.unipi.dii.activitiesexample

Source Language: Java

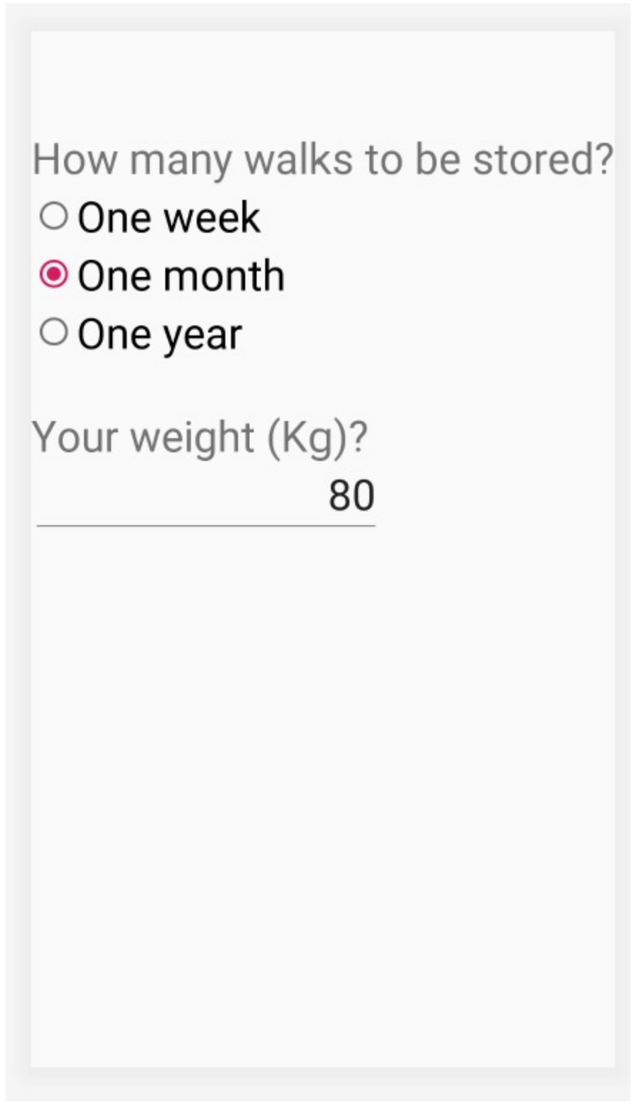
The name of the activity class to create

Define second activity

- Design the layout of the second activity and add String resources


Component Tree
ConstraintLayout
radioGroup2(vertical)
radioButton- "@string/opti...
radioButton2- "@string/opti..
radioButton3- "@string/opti..
textView2- "@string/field1"
textView3- "@string/weight"
editText- "80"

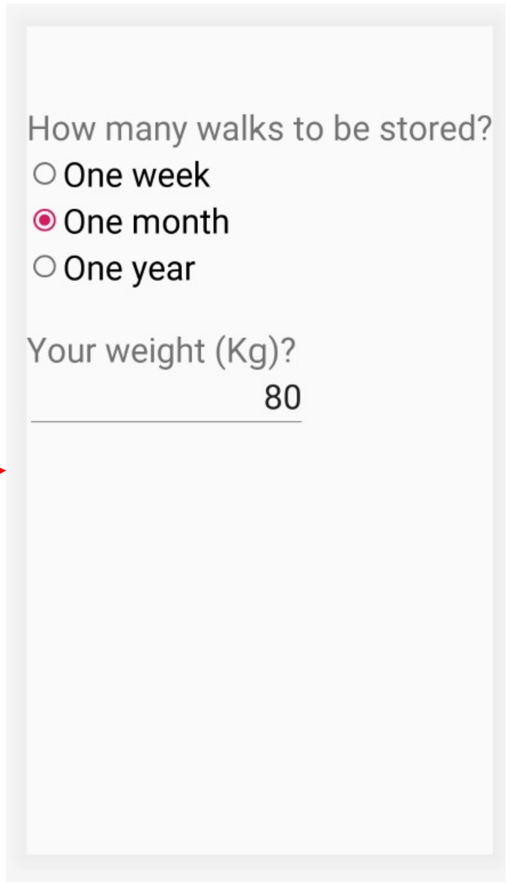
```
<resources>
    <string name="app_name">ActivitiesExample</string>
    <string name="button1_string">Your walks</string>
    <string name="button2_string">Settings</string>
    <string name="field1">How many walks to be stored?</string>
    <string name="option1">One week</string>
    <string name="option2">One month</string>
    <string name="option3">One year</string>
    <string name="weight">Your weight (Kg)?</string>
</resources>
```


How many walks to be stored?
One week [✓ One month] One year
Your weight (Kg)?
80

Activities in manifest file

- Android Studio automatically adds the Activity to the manisfest file

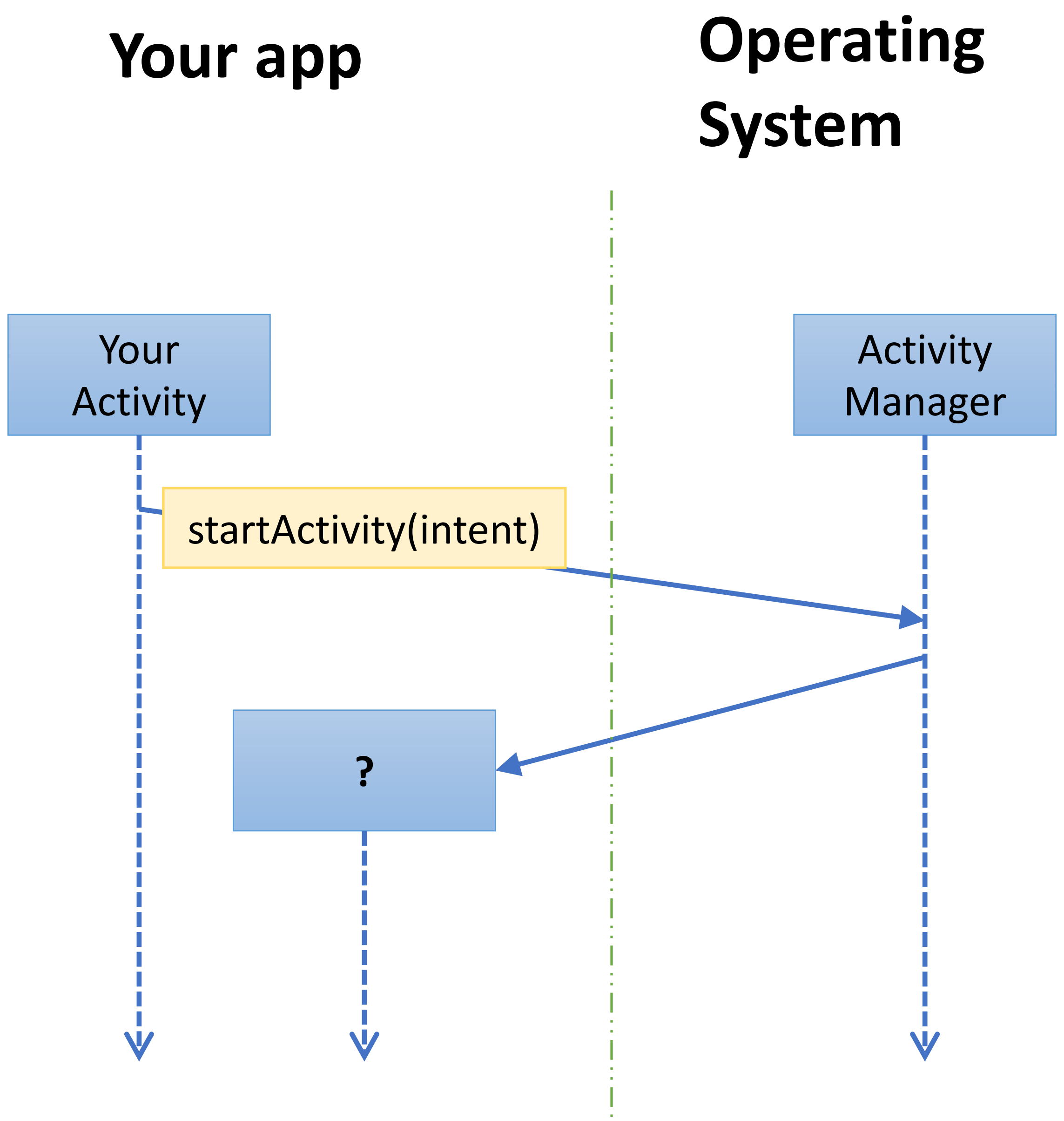
```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".SettingsActivity"></activity>
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```


How many walks to be stored?
One week [✓ One month] One year

Your weight (Kg)?
80

Starting another Activity

- Activity 1 starts Activity 2
  - through the Android OS
  - by calling *startActivity(Intent)*
- Passes *Intent* (object for communicating with Android OS)
- Intent specifies which (target) Activity the *ActivityManager* should start

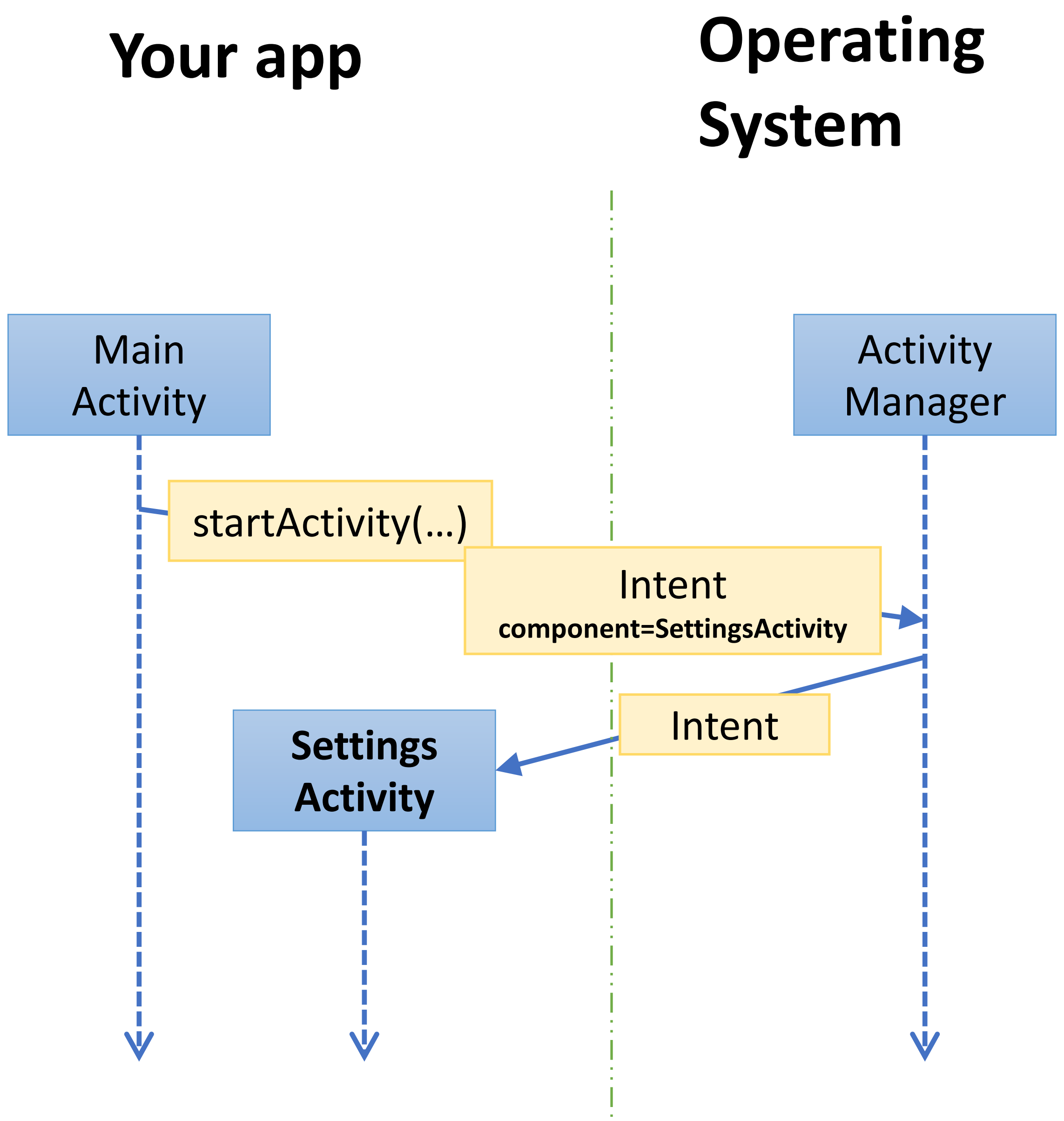


Starting another activity

- Intents have many different constructors. We will use this one:

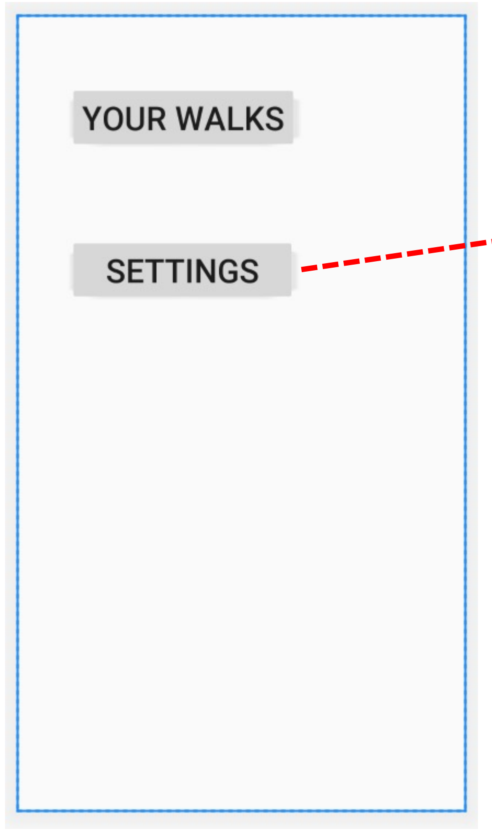
```
Intent(Context ctx, Class<?> cls)
```

- *Context*: the environment for the Intent, the starting activity
- *Class*: the activity to be started



Starting another activity

- Actual code:

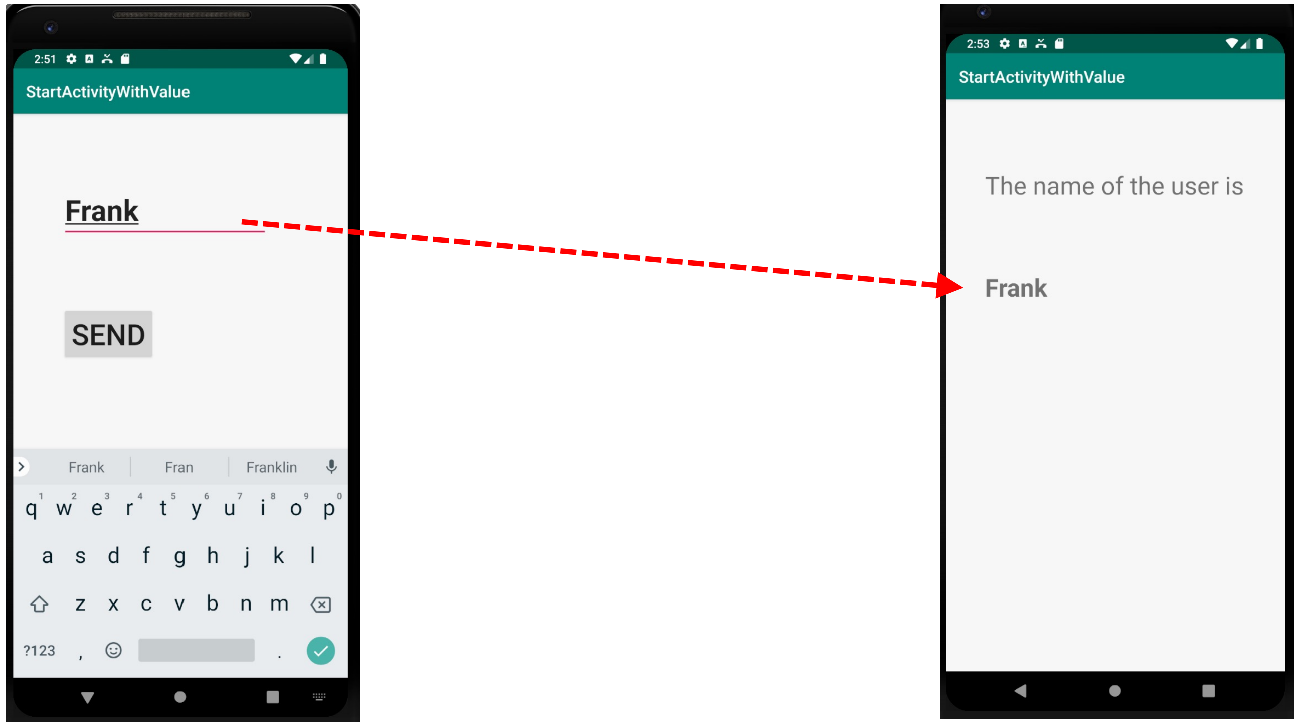


```
. . .
<Button
    android:id="@+id/button2"
    . . .
    android:onClick="startSettings"
    android:text="@string/button2_string"
/>
. . .
```

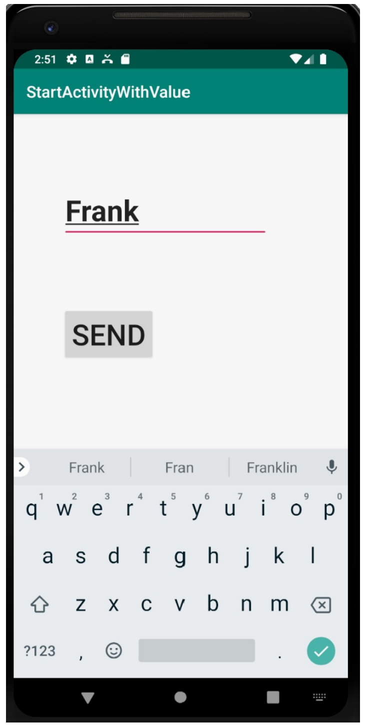
```
public class MainActivity extends AppCompatActivity {

    . . .

    public void startSettings(View v) {
        Intent i = new Intent(this, SettingsActivity.class);
        startActivity(i);
    }
}
```

Providing values to the started activity

- It is possible to provide some values to the activity that is going to be started
- Information is transferred as <*key, value*> pairs attached to intents



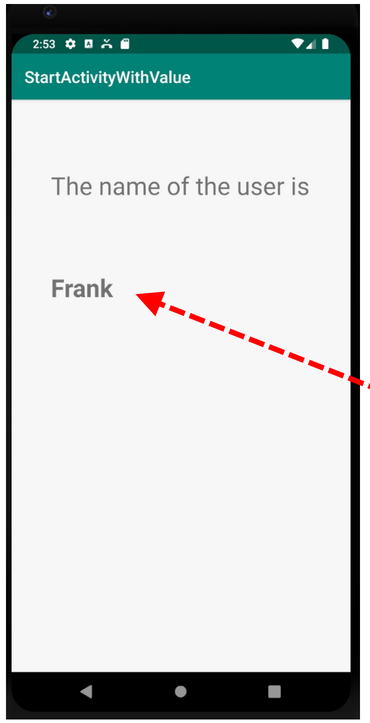
Providing values to the started activity



```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send"
    android:textSize="36sp"
    android:onClick="send"
    . . .
/>
```

```
public class MainActivity extends AppCompatActivity {

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

    public void send(View v) {
        Intent i = new Intent(this, ReceiverActivity.class);
        EditText et = (EditText) findViewById(R.id.editText);
        i.putExtra("user_name", et.getText().toString());
        startActivity(i);
    }
}
```

Providing values to the started activity


2:53

StartActivityWithValue

The name of the user is

Frank

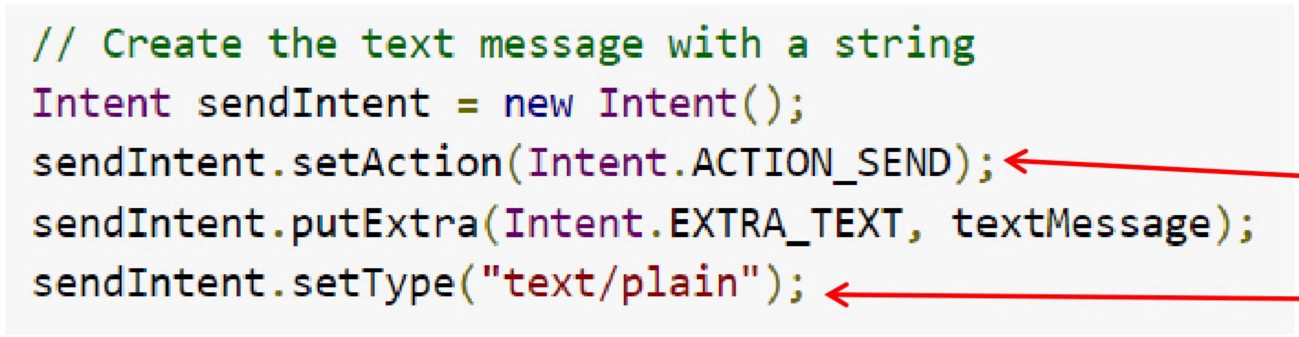
```
public class ReceiverActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receiver);
        Intent i = getIntent();
        String name = i.getStringExtra("user_name");
        TextView tv = (TextView) findViewById(R.id.textView2);
        tv.setText(name);
    }
}
```

Adding extras to an intent

- **user_name** is passed as extra on the *Intent* passed into *startActivity()*
- Extras are arbitrary data calling activity can include with intent
- To add extra to Intent, use *putExtra()* methods
- Different types of simple values are supported (int, float, string, etc)
- On the receiver side they can be retrieved using *getXXXExtra()* methods (e.g. *getStringExtra()*, *getIntExtra()*, etc)
- The same approach can be followed to provide a return value to the starting activity

Explicit and implicit intents

- Previous example: an **explicit** intent
  - The two activities are in the same app
- If the activity to be started is in another app, an **implicit** intent has to be used instead
- **Implicit intent**: it <u>does not</u> name component to start
- Specifies
  - **Action** (what to do, example visit a web page)
  - **Data** (to perform operation on, e.g. web page url)
- System decides component to receive intent based on **action**, **data**, **category**



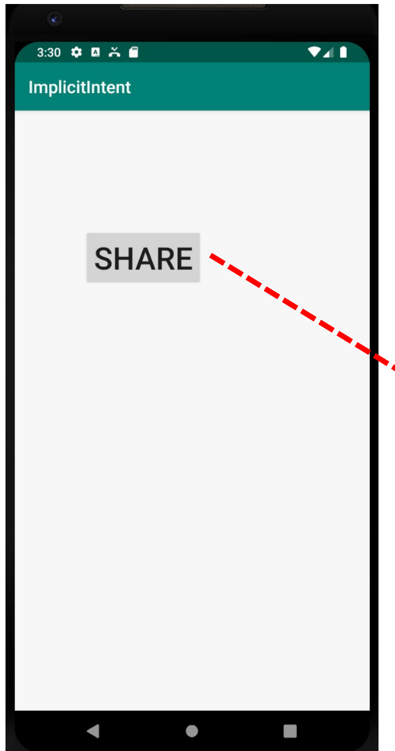
```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
```

ACTION (No receiving Activity specified)

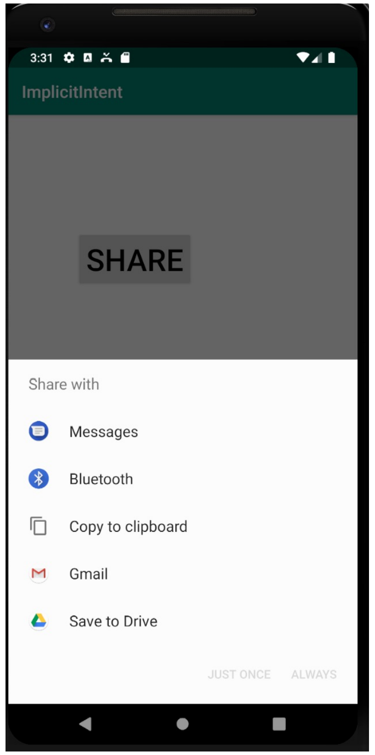
Data type

Implicit intents

- Typically, many components (apps) can carry out a given action
  - E.g. Many phones have installed multiple apps that can view images
- In this case Android shows a *Chooser*

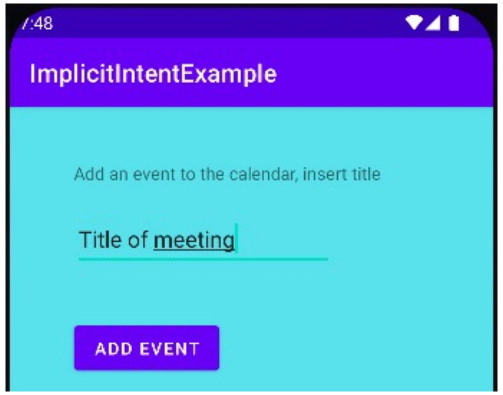


```
public void sendImplicitIntent(View v) {
    Intent i = new Intent(Intent.ACTION_SEND);
    i.setType("text/plain");
    i.putExtra(Intent.EXTRA_TEXT, "Text to be sent");
    startActivity(i);
}
```



Examples of implicit intents

- Setting an alarm
- Add a calendar event
- Take a picture
- View/edit a contact
- Send an email
- Show a location on the map
- Start a phone call
- Open settings

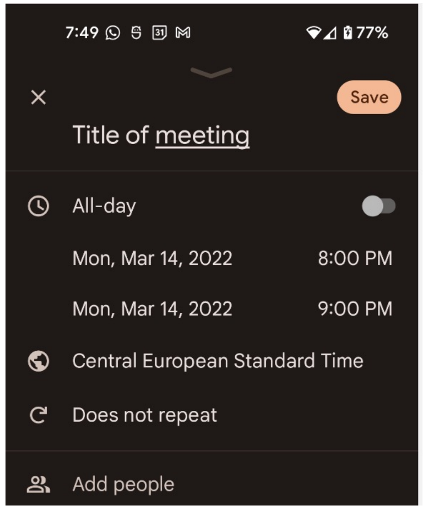


7:48

ImplicitIntentExample

Add an event to the calendar, insert title

Title of meeting



7:49 77%

Title of meeting

All-day

Mon, Mar 14, 2022 8:00 PM

Mon, Mar 14, 2022 9:00 PM

Central European Standard Time

Does not repeat

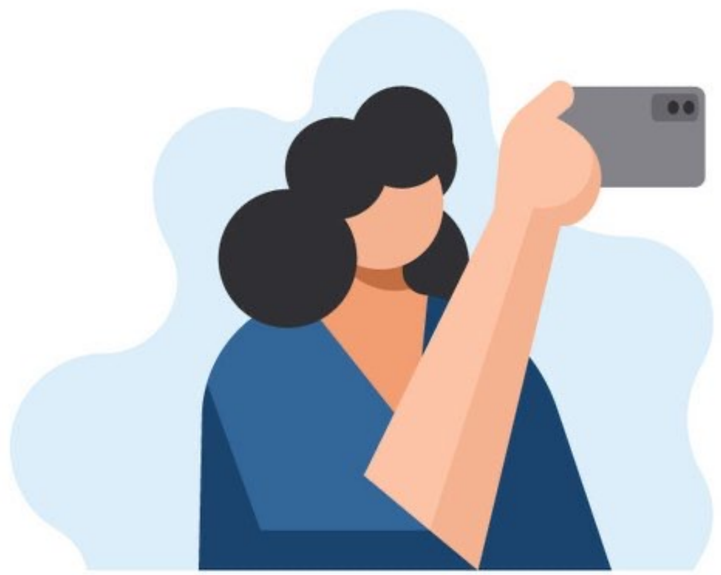
Add people

```
public void m(View v) {
    String title = ((EditText)findViewById(R.id.ed1)).getText().toString();
    String location = "Central Perk's";
    Intent intent = new Intent(Intent.ACTION_INSERT)
            .setData(CalendarContract.Events.CONTENT_URI)
            .putExtra(CalendarContract.Events.TITLE, title)
            .putExtra(CalendarContract.Events.DESCRIPTION, "Some coffee")
            .putExtra(CalendarContract.Events.EVENT_LOCATION, location)
            .putExtra(CalendarContract.Events.ALL_DAY, "true");
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

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

Taking pictures

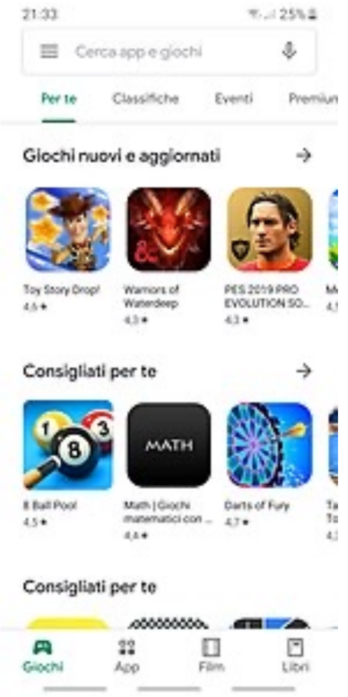
- Ref: https://developer.android.com/training/camera/photobasics.html
- How to take photos from your app using Android Camera app
- 4 Steps:
  - Request the camera feature
  - Take a Photo with the Camera App
  - Get the Thumbnail
  - Save the Full-size Photo



Camera feature

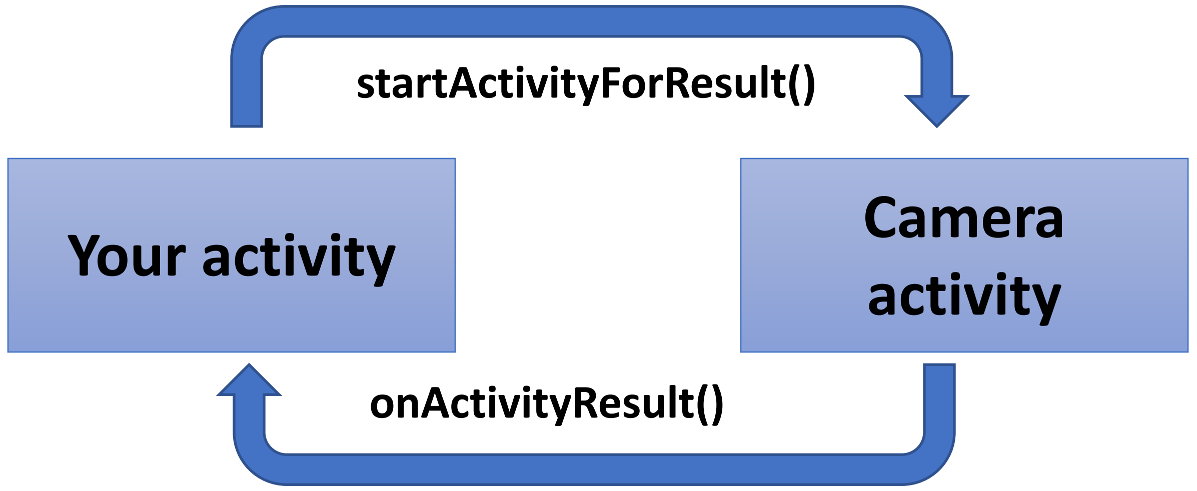
- If your app takes pictures using the phone’s Camera, you can allow only devices with a camera find your app while searching Google Play Store
- How? Make the following declaration in *AndroidManifest.xml*

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera"
                  android:required="true" />
    ...
</manifest>
```

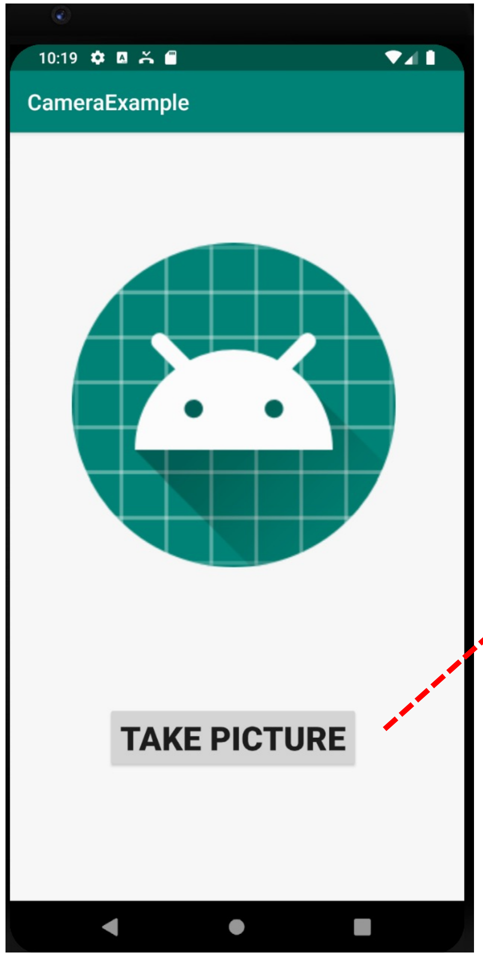


Capture an image with Camera app

- https://developer.android.com/training/camera/photobasics.html
- To take picture, your app needs to send implicit Intent requesting for a picture to be taken (i.e. action = capture an image)
- Call *startActivityForResult()* since picture is sent back with result intent
- Potentially, multiple apps/activities can handle this operation (take a picture)
- Check that at least one Activity can handle request to take picture using *resolveActivity()*

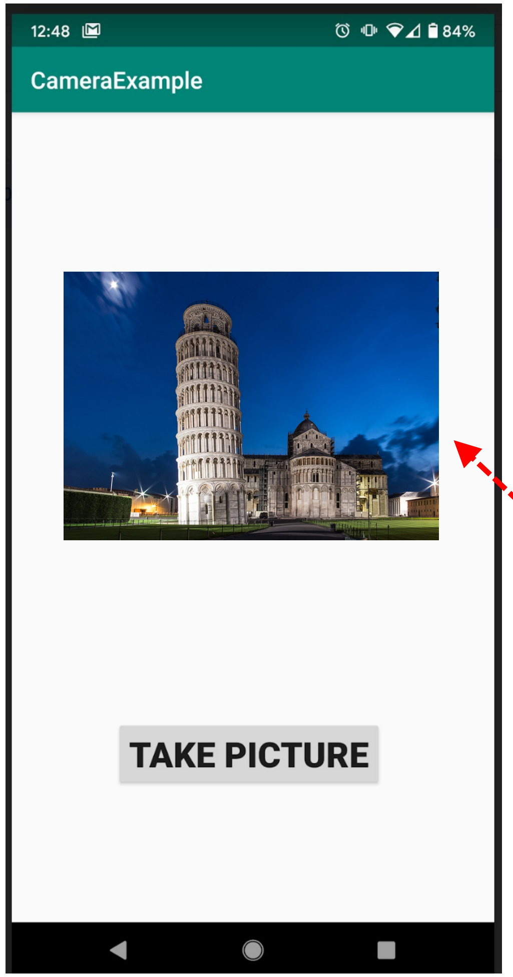


Capture an image with Camera app



```
…
private static int REQUEST = 1;

public void takePicture(View v) {
    Intent tp = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if(tp.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(tp, REQUEST);
    }
}
…
```

Capture an image with Camera app



- Thumbnail is returned by camera app
- Bitmap returned in “extra” of Intent delivered to *onActivityResult()*

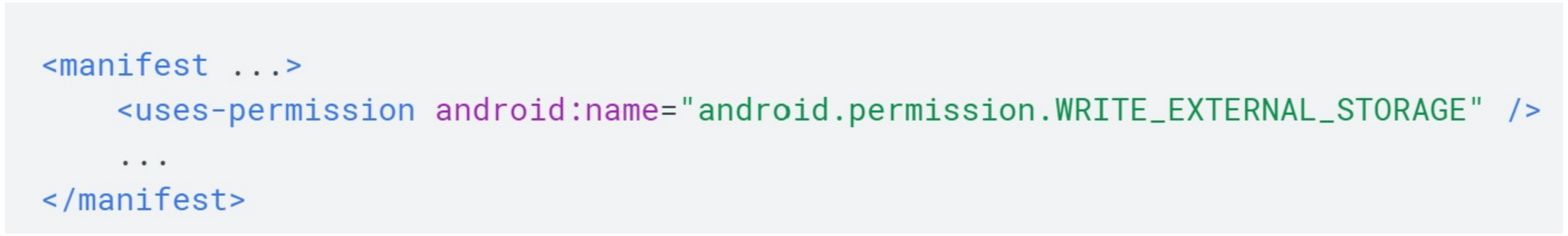
```
@Override
protected void onActivityResult(int requestCode, int resultCode,
                                                @Nullable Intent data) {
    if(requestCode == REQUEST && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap bm = (Bitmap) extras.get("data");
        ImageView iv = (ImageView) findViewById(R.id.imageView);
        iv.setImageBitmap(bm);
    }
}
```

Save full-sized photo

- https://developer.android.com/training/basics/data-storage/files.html
- Android Camera app saves full-sized photo in a filename you give it
- We need owner’s permission to write to external storage
- Android systems have:
    - **App specific storage**: data stored here is available by only your app
    - **Shared storage**: content can be available to other apps
- We would like all apps to see pictures this app takes, so use shared storage

Full-size pictures

- Android Camera app can save full-size photo to
  - Public external storage (shared by all apps)
    - *getExternalStoragePublicDirectory()*
    - Need to get permission
  - Private storage (seen by only your app, deleted when your app uninstalls):
    - *getExternalFilesDir()*
- Need phone owner’s permission to write to external storage
- In *AndroidManifest.xml* add the following declaration



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

Full-size pictures

- Some extra steps are required:
  - Setup a *FileProvider* in manifest file
  - Define properly the path where image is stored

  See docs

```
static final int REQUEST = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile(); // A method that generates filename
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
                                                      "it.unipi.dii.cameraexample",
                                                      photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            startActivityForResult(takePictureIntent, REQUEST);
        }
    }
}
```

Broadcast intents

- Some intents are broadcasted by Android to notify apps of system events
- Examples
  - Boot phase completed
  - Wi-Fi state changed
  - Timezone changed
  - SMS received
  - Power cord connected/disconnected
  - Power saving mode activated/deactivated
  - …

- For a complete list of system broadcast actions, see the BROADCAST_ACTIONS.TXT file in the Android SDK

BroadcastReceiver

- This broadcast receiver is activated when an SMS is received:

```java
public class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d("MY_RECEIVER", "I just received an SMS...");
    }
}
```

This method is executed by an OS thread: if long running operation you have to start a service

BroadcastReceiver

- To receive events, a *BroadcastReceiver* must be registered
- Two options:
  - **Static**: in the manifest file, receives events even if app is not running
  - **Dynamic**: by means of Java code, e.g. in Activities

```
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

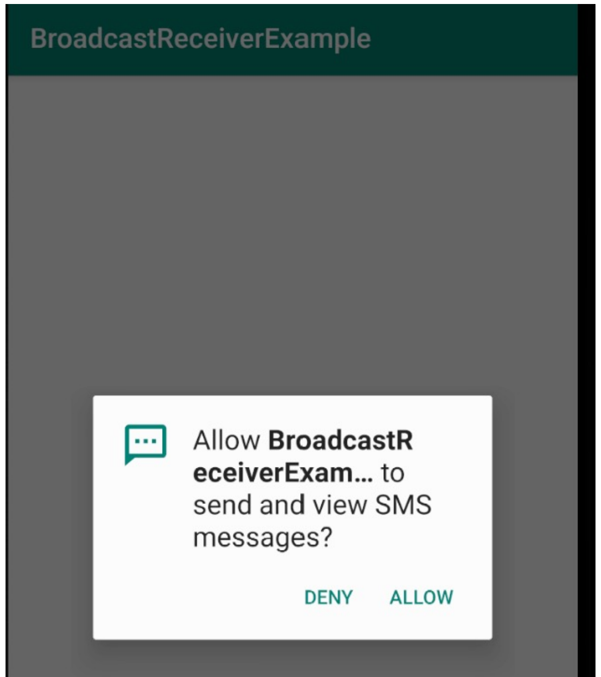
Filter limits the type of intents that will be received

BradcastReceiver

- These event requires permissions
- In the manifest:

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

- Permissions will be requested to the user when app is executed
- Permission can also be requested through Java code
- The number of system broadcasts has been reduced in Android 7+ for improved energy and memory efficiency


BroadcastReceiverExample

Allow **BroadcastReceiverExam...** to send and view SMS messages?

Runtime permissions

- In Android 6.0+ must be requested at runtime (not installation time)
- Requesting permission needed for the SMS receiver example:

```java
private void requestSmsPermission() {
    Log.i("BroadcastReceiverExample", "Requesting SMS permission");
    String permission = Manifest.permission.RECEIVE_SMS;
    int grant = ContextCompat.checkSelfPermission(this, permission);
    if (grant != PackageManager.PERMISSION_GRANTED) {
        String[] permissionList = new String[1];
        permissionList[0] = permission;
        ActivityCompat.requestPermissions(this, permissionList, 1);
    }
}
```

BroadcastReceiver: dynamic registration/unregistration

- The same receiver can be registered and unregistered dynamically

```
private static String TAG = "BroadcastReceiverExample";
MyReceiver mr = new MyReceiver();

@Override
protected void onResume() {
    super.onResume();
    IntentFilter filter = new IntentFilter();
    filter.addAction(Telephony.Sms.Intents.SMS_RECEIVED_ACTION);
    registerReceiver(mr, filter);
    Log.i(TAG, "Registering receiver");
}

@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mr);
    Log.i(TAG, "Unregistering receiver");
}
```

References

- CS 528 Mobile and Ubiquitous Computing, WPI
- Android Big Nerd Ranch 3rd edition
- https://developer.android.com
- https://developer.android.com/training/basics/data-storage/files.html
- https://developer.android.com/training/camera/photobasics.html