Outline

- Intents and intent filters
- Explicit and implicit intents
- Activity lifecycle
- Apps with multiple activities

Intent

- A messaging object to activate another app component
  - Communicate between an app and the host system
- Facilitate interactions and collaboration among components
  - Start an activity
  - State a service
  - Deliver a broadcast message
  - Deliver data
Intent Types

- **Explicit intents**
  - Explicitly name the component to be started
  - Typically used to start a component within your own app
- **Implicit intents**
  - Do not name a specific component to be started
  - Indicate a general action to be performed
  - Allow another app, the receiver, to choose a suitable component to perform the action
    - e.g., to request another app to show a specified location on a map.

Intent Filter

- Declares the capabilities or intentions of an app of handling certain types of actions
- An expression in the app's manifest file
  - Specifies the type of **Intents** that the component may receive or is capable to respond
  - Makes it possible for other apps to start an activity in your app with an **implicit intent**.
  - If no intent filter is declared for an activity, then it can be started only with an **explicit intent**.

Start an Activity

- An **Intent** object carries several types of information
  - When used as an explicit intent
    - **Component name** – The name of the component to start.
    - **Extras** – Key-value pairs that carry additional
  - Start an activity with an explicit intent
    - in the context of an activity

```java
Intent intent = new Intent(this, TargetActivity.class);
startActivity(intent);
```

Android Activity Lifecycle
Android Activity

- A single, focused thing that the user can do.
  - Interact with the user
- Often presented as a full-screen window
- Can also be presented as
  - Floating windows, or
  - Embedded inside of another activity
- One of the smallest unit that can be independently created, paused, resumed, and destroyed

Activity Lifecycle States

- An activity can be in one of the following lifecycle states:
  - **Active**: an activity is in the foreground, at the top of the activity stack.
  - **Paused**: an activity has lost focus but is still visible (non-full-sized activity).
  - **Stopped**: an activity is completely obscured by another activity.
  - **Destroyed**: an activity killed by the system.

Activity Lifecycle Methods

- *Activity lifecycle methods* are callback methods invoked by the systems during the transition between lifecycle states of an activity.

```java
protected void onCreate(Bundle savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
```
Activity Lifecycle States

- **Foreground, top of the stack**
- **Visible**
  - Retain resources, **Killable (pre-Honeycomb)**

Activity Lifecycle Callbacks
- **Creation**
  - Called when the activity is first created.
  - Called when the activity is becoming visible.

Activity Lifecycle Callbacks
- **Going into Background**
  - Called when the system is about to start resuming a previous activity.
  - Called when the activity is no longer visible to the user.
  - The final call you receive before your activity is destroyed.

Activity Lifecycle Callbacks
- **Pausing and Resumption**
  - Called when the activity will start interacting with the user.
  - Reinitialize/revert the effects of onPause().
  - Called when the system is about to start resuming a previous activity.
  - Stop animation.
  - Commit unsaved changes.
  - Release unused resources.
Activity Lifecycle Callbacks – Stopping and Restarting

- **Called when the activity will start interacting with the user.**
- **Called when the activity is becoming visible. Reinitialize**
- **Called when the activity is no longer visible to the user. Release unused resources**
- **Called after your activity has been stopped, prior to it being started again.**

Activity Lifecycle Callbacks – Recreating an Activity

- **Called before the activity is killed.**
- **Persist the activity state**
- **Called when the activity is being re-initialized from a previously saved state. Restore the activity state from previously saved state**
- **Called after your activity has been started again.**

Tasks and the Back Stack

- An app usually contains multiple activities.
  - May even start activities in other apps
- A **task** is a collection of activities that users interact with when performing a certain job.
- The activities are arranged in a stack (the "back stack"), in the order in which each activity is opened.
  - Strictly LIFO ("last in, first out")
  - Push: launch an app from Home screen, or start an activity from another activity
  - Pop: Back button tapped, the current activity is popped
The Back Stack

Activities and Tasks

- Activity A starts Activity B
  - Activity A is stopped, and its state is retained.
  - Activity B is pushed onto the top of the stack.
- Tap the Back button,
  - Activity B is popped and destroyed, and its state is not retained
  - Activity A resumes with its state restored.

Background Tasks

- Tap the Home button, leaves a task
  - The current activity is stopped.
  - The current task goes into the background
  - The state of every activity in the task is retained
- The task can be resumed through
  - Launcher icon on Home screen, or
  - The Recent Tasks

Background Tasks
Multiple Instances of Activities

- Activities can be instantiated multiple times
  - Multiple instances of the same activity may exist in the back stack

Intent Demo App

- An app with two screens, i.e., two activities
  - **Activity A** can
    - Start **Activity B**, with an explicit intent
    - Finish itself
  - **Activity B** can
    - Start **Activity A**, with an explicit intent
    - Finish itself

Building the Intent Demo App – Adding a New Activity

- New Project “Intent Demo”
  - Start with a blank activity
  - Change Activity name to: “ActivityA”
  - Change Layout name to: “activity_a”
- File | New ...
  - **Activity | Blank Activity**
  - Choose Activity name: “ActivityB”
  - Choose Layout name: “activity_b”
  - Do not check “launcher activity”
  - Finish
The Android Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.depaul.csc472.intentdemo">
    <application android:allowBackup="true" android:icon="@drawable/ic_launcher" android:label="IntentDemo" android:theme="@style/AppTheme">
        <activity android:name=".ActivityA"
                android:label="@string/title_a">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".ActivityB"
                android:label="@string/title_b">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Activity names should be fully qualified. ’.’ – the current package.

The Layout of the Screens

- Layout file: `activity_a.xml`

```xml
<LinearLayout
    ...
    <TextView
        android:id="@+id/title"
        android:text="Activity A"/>
    ...
    
    <Button
        android:id="@+id/button1"
        android:text="Start Activity B"/>
    ...
    ...
</LinearLayout>
```

- Layout file: `activity_b.xml` is nearly identical

Keep Track of Activities – `onCreate` and `onStart`

```java
public class ActivityA extends Activity {
    private static int INSTANCE_COUNTER = 0;
    private int instanceID;
    private int counter = 0;

    public ActivityA() {
        super();
        Log.d(TAG, "Constructor");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_a);
        Intent intent = new Intent(ActivityA.this, ActivityB.class);
        startActivity(intent);
    }
}
```

Start a New Activity – `ActivityA.java`

```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_a);
    Button button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(ActivityA.this, ActivityB.class);
            startActivity(intent);
        }
    });
    ...
    ...
    
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() counter=" + ++counter);
    }
    ...
    ...
```
Finish the Current Activity
– ActivityA.java

```java
protected void onCreate(Bundle savedInstanceState) {
    Button button2 = (Button) findViewById(R.id.button2);
    button2.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View view)
        {
            finish();
        }
    });
}
```
The Intent Demo

1. Launch the demo
   - ActivityA
2. Start Activity
   - ActivityB
3. Start Activity
   - A new instance of ActivityA

Tip: To Rename a File

- Do not manually change file names.
- Use the Refactoring feature of Android Studio
  - It will search for references to the file in your project, and will replace all the references with the new name
- Select the file, or other item, that you want to rename in the navigator, then
  - Right-click (Ctrl-Click on Mac) or
  - From the menu bar
  - Refactor | Rename

Passing Data Between Activities

1. Launch the demo
   - ActivityA
2. Start Activity
   - ActivityB
3. Start Activity
   - A new instance of ActivityA
4. Finish
   - Pop ActivityA
   - ActivityB counter incremented
The Bundle Class

- **Class:** `android.os.Bundle`
- A mapping from `String` keys to various `Parcelable` values.
- Used extensively in Android for passing and saving data.

```java
protected void onCreate(Bundle savedInstanceState) {
  // A Bundle object representing the state of the Activity in its previous incarnation
}
```

What is Parcelable?

Class: `android.os.Parcelable`

- Representing data that can be written to and restored from a `Parcel`
- A `Parcel` is a container of data that can be flattened and unflattened, or marshalled and unmarshalled

```
class Data {
  void method(int x, int y) {
    // some code...
  }
}
```

Serializable vs. Parcelable

- **Serializable** is a Java standard mechanism
  - Simple. A marker interface, no methods.
  - Slow. JVM does the work for you. Use reflection!
- **Parcelable** is an Android specific mechanism
  - Common types are `Parcelable`.
  - More code for user-defined classes.
  - Developers are responsible for handling the reading and writing of data
  - Much faster! As much as 10x improvement.

Get Results from Another Activity

- Start an activity and expect a result
  - an explicit intent and a request code (int)
    ```java
    Intent intent = new Intent(this, TargetActivity.class);
    startActivityForResult(intent, req_code);
    ```
  - Implement a callback
    ```java
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
      if (requestCode == req_code) {
        if (resultCode == RESULT_OK) { ... }
      }
    }
    ```
```

```
Marshall/Flatten

1001110 ... 1001000

Unmarshall/Unflatten
```

```java
class Data {
  void method(int x, int y) {
    // some code...
  }
}
```
Return Results to the Caller

- Set the result that your activity will return to its caller.
- Normal return, provide the result in an intent

```java
Intent data = new Intent();
data.putExtra(key, value);
setResult(RESULT_OK, data);
```

- Cancel

```java
setResult(RESULT_CANCELED);
```

Storing Data in and Retrieving Data from an Intent

- An Intent object may carry extra data
- Store extra data in an intent

```java
intent.putExtra(key, value);
```

- key is a string
- value can be any Parcelable data, including, primitive types, arrays, and strings
- Retrieve data from an intent

```java
intent.getStringExtra(key);
```

- Type is the name of the type of the value to be retrieved
- e.g., getFloatExtra(key); getStringExtra(key);

Intent + Data Demo App

Who Wants to be a Millionaire? – Ask a Friend

- An app with two activities
- My Activity
  - Asks a question
  - Starts the My Friend Activity, pass the question as the data
- My Friend Activity
  - Retrieve the question
  - Give an answer, send back to My Activity
  - Finish
My Activity – The Layout

```xml
<LinearLayout ...
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView ...
        android:text="What is your question" />
    <EditText ...
        android:id="@+id/question"
        android:hint="type a question" />
    <Button ...
        android:id="@+id/ask"
        android:text="Ask a friend"
        android:layout_gravity="right" />
</LinearLayout>
```

My Activity – Ask a Friend

```java
public class MyActivity extends Activity {
    private static final int ASK_QUESTION = 100; // request code

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
        Button button = (Button) findViewById(R.id.ask);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MyActivity.this, MyFriendActivity.class);
                EditText question = (EditText) findViewById(R.id.question);
                intent.putExtra("Question", question.getText());
                startActivityForResult(intent, ASK_QUESTION);
            }
        });
    }
}
```

Pass the question as an extra in the intent.

My Friend Activity – The Layout

```xml
<LinearLayout ...
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView ...
    <TextView ...
    <EditText ...
        android:id="@+id/answer" />
    <LinearLayout ...
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Button ...
            android:id="@+id/dontknow"
            android:text="I don't know" />
        <Button ...
            android:id="@+id/reply"
            android:text="Reply" />
    </LinearLayout>
</LinearLayout>
```

My Friend Activity – Show Me the Question

```java
@override
protected void onStart() {
    Intent intent = getIntent();
    if (intent != null) {
        TextView question = (TextView) findViewById(R.id.question);
        question.setText(intent.getStringExtra("Question"));
    }
}
```

What is CharSequence?
Char Sequence is an interface in Java
- java.lang.CharSequence
- It represents a sequence of characters
- Java String class implements the Char Sequence interface
- Android API uses Char Sequence extensively, where you would normally expect String
- It allows different classes that implement the Char Sequence interface to be used
- For better performance, Android framework internally uses an alternative implementation of Char Sequence in lieu of String

My Friend Activity
– Replay to the Question

```java
protected void onCreate(Bundle savedInstanceState) {
    Button reply = (Button) findViewById(R.id.reply);
    reply.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            EditText answer = (EditText) findViewById(R.id.answer);
            Intent data = new Intent();
            data.putExtra("Answer", answer.getText());
            setResult(RESULT_OK, data);
            finish();
        }
    });
}
```
Intent + Data Demo

- **My Activity**
  - Type a question

- **My Friend Activity**
  - Type an answer, “Reply” or
  - “I don’t know”

---

Implicit Intents

- **My Activity**
  - Type a question

- **My Friend Activity**
  - Type an answer, “Reply” or
  - “I don’t know”

- **My Activity**
  - Receive the answer, or
  - Receive “I don’t know”
Delivering an Implicit Intent

1. **Activity A** creates an **Intent** with an action description and passes it to `startActivity()`.
2. The Android System searches all apps for an intent filter that matches the intent. When a match is found.
3. The system starts the matching activity (**Activity B**) by invoking its `onCreate()` method and passing it the **Intent**.

### Code Example

```java
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

Delivering an Implicit Intent

- It is possible that a user doesn’t have any app that can handle the **Intent** sent in `startActivity()`.
- If this happens, the call will fail and the app will crash.
- To ensure that an **Intent** can be handled by some activity.
Building Implicit Intents

- Instead of specifying a component name, specifying an action
  - e.g., ACTION_VIEW, ACTION_SEARCH, ACTION_SEND
- Set data
  - A Uri object
    - Define a URI reference, e.g., http://www.android.com
    - Method: URI.parse\(\text{string}\)\)
- Set extras
  - key-value pairs of additional data

Implicit Intent Demo

- A simple app with button
- Each button invokes an action that uses an implicit intent to activate another app on your device or emulator
  - Open a web page
  - Perform a web search
  - Find a location on a map
  - Dial a phone number
  - Send an email

URI Syntax

- URI – Universal Resource Identifier
  - An Internet standard (RFC 3986)
  - URL is a special case of URI
- URI scheme
  - scheme-name : hierarchical-part [ ? query ] [ # fragment ]
    - Scheme name: http: tel:
    - Hierarchical part: d.android.com/guide/ui
    - Query: ?key1=value1;key2=value2
    - Fragment: #section2

Implicit Intent Demo – Open Web Page

- Action: ACTION_VIEW
- Data URI Scheme
  - http:\<URL>  
  - https:\<URL>

```java
public void openWebPage(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("http://www.android.com"));
    startActivity(intent);
}
```
Implicit Intent Demo – Search Web

- Action: ACTION_SEARCH
- Extras
  - SearchManager.QUERY
  - The search string.

```java
public void searchWeb(View view) {
    Intent intent =
        new Intent(Intent.ACTION_SEARCH);
    intent.putExtra(SearchManager.QUERY, "android watch");
    startActivity(intent);
}
```

Implicit Intent Demo – Show Map Location

- Action: ACTION_VIEW
- Data URI Scheme
  - geo:lat,lng
  - geo:lat,lng?z=zoom
  - geo:0.0?q=my+street+address

```java
public void showMap(View view) {
    Intent intent =
        new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse(  
        "geo:0,0?q=243+South+Wabash+Ave+Chicago+IL"));
    startActivity(intent);
}
```

Implicit Intent Demo – Dial Phone

- Action:
  - ACTION_DIAL – Opens the dialer or phone app.
  - ACTION_CALL – Places a phone call (requires the CALL_PHONE permission)
- Data URI Scheme
  - tel:<phone-number>

```java
public void dial(View view) {
    Intent intent =
        new Intent(Intent.ACTION_DIAL);
    startActivity(intent);
}
```

Implicit Intent Demo – Email

- Action:
  - ACTION_SENDTO – for no attachment.
  - ACTION_SEND – for one attachment
  - ACTION_SEND_MULTIPLE – for multiple attachments
- Extras
  - Intent.EXTRA_EMAIL – recipients
  - Intent.EXTRA_SUBJECT – email subject
  - Intent.EXTRA_TEXT – email body
Implicit Intent Demo – Email

```java
public void email(View view) {
    Intent intent =
        new Intent(Intent.ACTION_SEND);
    intent.setType("*/*");
    intent.putExtra(Intent.EXTRA_EMAIL,
           new String[] { "me@cdm.depaul.edu" });
    intent.putExtra(Intent.EXTRA_SUBJECT,
           "Email from Android App");
    intent.putExtra(Intent.EXTRA_TEXT,
           "-- Send by Android App");
    startActivity(intent);
}
```

The Sample Code

- The sample apps in this lecture are available in D2L
  - IntentDemo.zip
  - Intent+DataDemo.zip
  - ImplicitIntentDemo.zip
- Each zip archive contains the entire project folder
- Unzip the file and import to Android Studio
- The sample project minimum SDK version is set to L-preview
- You may change the minSdkVersion in build.gradle

Next ...

- Adapters and Adapter Views
- Spinners
- List Views and List Activities
- Customize List Views and Adapters

Android is a trademark of Google Inc.
Some contents and diagrams are from Google Inc.
Licensed under Apache 2.0 and Creative Commons Attribution 2.5.