Android Developer Fundamentals V2

Activities and Intents

Lesson 2

2.3 Implicit Intents

Contents

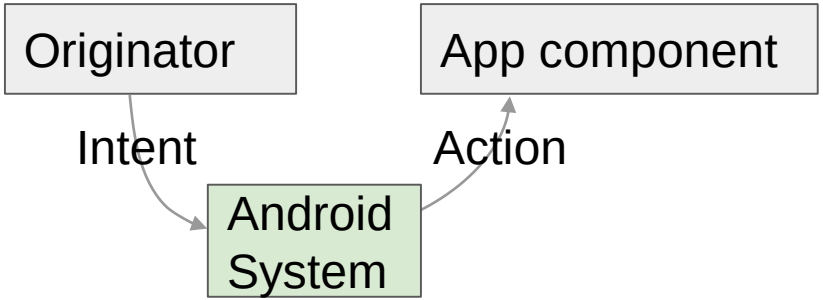
- Intent—recap
- Implicit Intent overview
- Sending an implicit Intent
- Receiving an implicit Intent

Recap: Intent

What is an Intent?

An `Intent` is:

- Description of an operation to be performed
- Messaging object used to request an action from another app component via the Android system.



What can an Intent do?

An `Intent` can be used to:

- start an Activity
- start a Service
- deliver a Broadcast

Services and Broadcasts are covered in other lessons

Explicit vs. implicit Intent

Explicit Intent — Starts an Activity of a specific class

Implicit Intent — Asks system to find an Activity class with a registered handler that can handle this request

Implicit Intent overview

What you do with an implicit Intent

- Start an Activity in another app by describing an action you intend to perform, such as "share an article", "view a map", or "take a picture"
- Specify an action and optionally provide data with which to perform the action
- Don't specify the target Activity class, just the intended action

What system does with implicit Intent

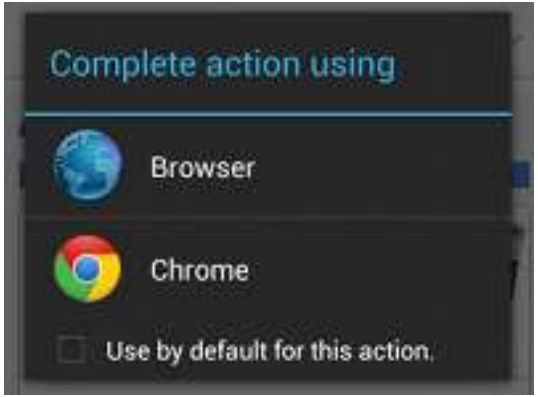
- Android runtime matches the implicit intent request with registered intent handlers
- If there are multiple matches, an App Chooser will open to let the user decide

How does implicit Intent work?

1. The Android Runtime keeps a list of registered Apps
2. Apps have to register via AndroidManifest.xml
3. Runtime receives the request and looks for matches
4. Android runtime uses Intent filters for matching
5. If more than one match, shows a list of possible matches and lets the user choose one
6. Android runtime starts the requested activity

App Chooser

When the Android runtime finds multiple registered activities that can handle an implicit Intent, it displays an App Chooser to allow the user to select the handler



Sending an implicit Intent

Sending an implicit Intent

1. Create an Intent for an action

```
Intent intent = new Intent(Intent.ACTION_CALL_BUTTON);
```

User has pressed Call button — start Activity that can make a call (no data is passed in or returned)

1. Start the Activity

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

Avoid exceptions and crashes

Before starting an implicit Activity, use the package manager to check that there is a package with an Activity that matches the given criteria.

```
Intent myIntent = new Intent(Intent.ACTION_CALL_BUTTON);

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

Sending an implicit Intent with data URI

1. Create an Intent for action

```
Intent intent = new Intent(Intent.ACTION_DIAL);
```

1. Provide data as a URI

```
intent.setData(Uri.parse("tel:8005551234"));
```

1. Start the Activity

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

Providing the data as URI

Create an URI from a string using `Uri.parse(String uri)`

- `Uri.parse("tel:8005551234")`
- `Uri.parse("geo:0,0?q=brooklyn%20bridge%2C%20brooklyn%2C%20ny")`
- `Uri.parse("http://www.android.com");`

Uri documentation

Implicit Intent examples

Show 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);
```

Sending an implicit Intent with extras

1. Create an Intent for an action

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
```

1. Put extras

```
String query = edittext.getText().toString();
intent.putExtra(SearchManager.QUERY, query));
```

1. Start the Activity

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

Category

Additional information about the kind of component to handle the intent.

- `CATEGORY_OPENABLE`
  Only allow URIs of files that are openable
- `CATEGORY_BROWSABLE`
  Only an Activity that can start a web browser to display data referenced by the URI

Sending an implicit Intent with type and category

1. Create an Intent for an action

```
Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);
```

1. Set mime type and category for additional information

```
intent.setType("application/pdf"); // set MIME type
intent.addCategory(Intent.CATEGORY_OPENABLE);
```

continued on next slide...

Sending an implicit Intent with type and category

3. Start the Activity

```
if (intent.resolveActivity(getPackageManager()) != null) {
  startActivityForResult(myIntent,ACTIVITY_REQUEST_CREATE_FILE);
}
```

4. Process returned content URI in `onActivityResult()`

Common actions for an implicit Intent

Common actions include:

- ACTION_SET_ALARM
- ACTION_IMAGE_CAPTURE
- ACTION_CREATE_DOCUMENT
- ACTION_SENDTO
- and many more

Apps that handle common actions

Common actions are usually handled by installed apps (both system apps and other apps), such as:

- Alarm Clock, Calendar, Camera, Contacts
- Email, File Storage, Maps, Music/Video
- Notes, Phone, Search, Settings
- Text Messaging and Web Browsing

➔ List of common actions for an implicit intent

➔ List of all available actions

Receiving an Implicit Intent

Register your app to receive an Intent

- Declare one or more Intent filters for the Activity in AndroidManifest.xml
- Filter announces ability of Activity to accept an implicit Intent
- Filter puts conditions on the Intent that the Activity accepts

Intent filter in AndroidManifest.xml

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

Intent filters: action and category

- `action` — Match one or more action constants
  - `android.intent.action.VIEW` — matches any Intent with `ACTION_VIEW`
  - `android.intent.action.SEND` — matches any Intent with `ACTION_SEND`

- `category` — additional information (list of categories)
  - `android.intent.category.BROWSABLE`—can be started by web browser
  - `android.intent.category.LAUNCHER`—Show activity as launcher icon

Intent filters: data

- `data` — Filter on data URIs, MIME type
  - `android:scheme="https"`—require URIs to be https protocol
  - `android:host="developer.android.com"`—only accept an Intent from specified hosts
  - `android:mimeType="text/plain"`—limit the acceptable types of documents

An Activity can have multiple filters

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    ...
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    ...
  </intent-filter>
</activity>
```

An Activity can have several filters

A filter can have multiple actions & data

```
<intent-filter>
  <action android:name="android.intent.action.SEND"/>
  <action android:name="android.intent.action.SEND_MULTIPLE"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="image/*"/>
  <data android:mimeType="video/*"/>
</intent-filter>
```

Learn more

Learn more

- Intent class documentation
- Uri documentation
- List of common apps that respond to implicit intents
- List of available actions
- List of categories
- Intent Filters

What's Next?

- Concept Chapter: 2.3 Implicit Intents
- Practical: 2.3 Implicit Intents

END

Android Developer Fundamentals V2

Activities and Intents

Lesson 2

2.1 Activities and Intents

Contents

- Activities
- Defining an `Activity`
- Starting a new `Activity` with an `Intent`
- Passing data between activities with extras
- Navigating between activities

Activities (high-level view)

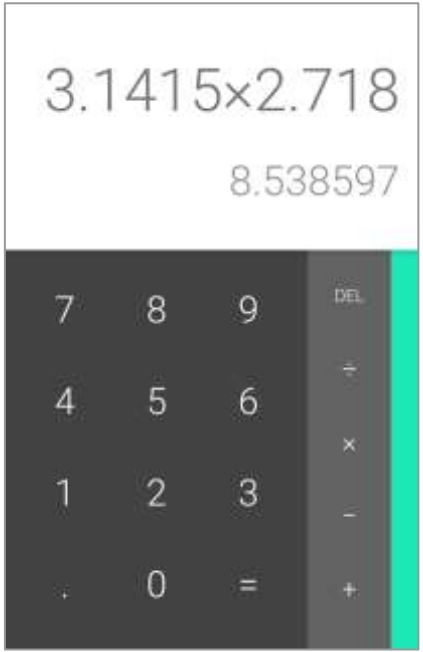
What is an Activity?

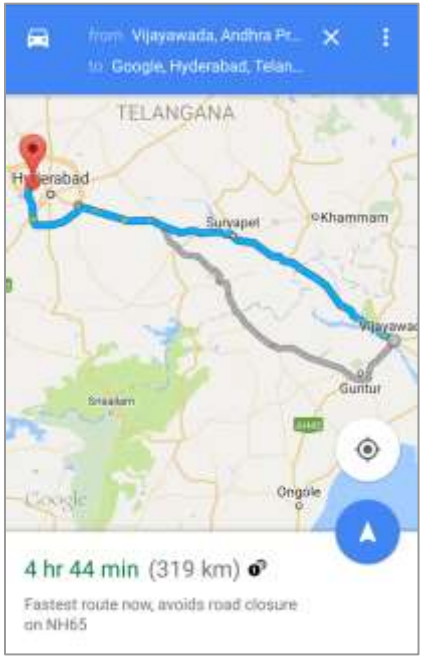
- An `Activity` is an application component
- Represents one window, one hierarchy of views
- Typically fills the screen, but can be embedded in other Activity or a appear as floating window
- Java class, typically one Activity in one file

What does an Activity do?

- Represents an activity, such as ordering groceries, sending email, or getting directions
- Handles user interactions, such as button clicks, text entry, or login verification
- Can start other activities in the same or other apps
- Has a life cycle—is created, started, runs, is paused, resumed, stopped, and destroyed

Examples of activities





Apps and activities

- Activities are loosely tied together to make up an app
- First Activity user sees is typically called "main activity"
- Activities can be organized in parent-child relationships in the Android manifest to aid navigation

Layouts and Activities

- An `Activity` typically has a UI layout
- Layout is usually defined in one or more XML files
- `Activity` "inflates" layout as part of being created

Implementing Activities

Implement new activities

1. Define layout in XML
2. Define `Activity` Java class
   - extends `AppCompatActivity`
3. Connect `Activity` with Layout
   - Set content view in `onCreate()`
4. Declare `Activity` in the Android manifest

1. Define layout in XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Let's Shop for Food!" />
</RelativeLayout>
```

2. Define Activity Java class

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

3. Connect activity with layout

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Resource is layout in this XML file

4. Declare activity in Android manifest

```
<activity android:name=".MainActivity">
```

4. Declare main activity in manifest

`MainActivity` needs to include `intent-filter` to start from launcher

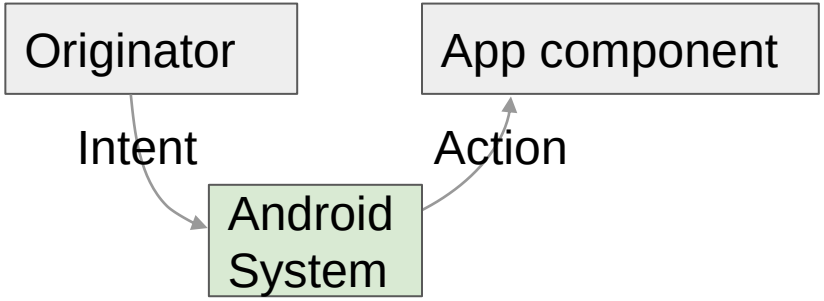
```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Intents

What is an intent?

An `Intent` is a description of an operation to be performed.

An `Intent` is an object used to request an action from another app component via the Android system.



What can intents do?

- Start an `Activity`
  - A button click starts a new `Activity` for text entry
  - Clicking Share opens an app that allows you to post a photo
- Start an `Service`
  - Initiate downloading a file in the background
- Deliver `Broadcast`
  - The system informs everybody that the phone is now charging

Explicit and implicit intents

Explicit Intent

- Starts a specific `Activity`
  - Request tea with milk delivered by Nikita
  - Main activity starts the `ViewShoppingCart Activity`

Implicit Intent

- Asks system to find an `Activity` that can handle this request
  - Find an open store that sells green tea
  - Clicking Share opens a chooser with a list of apps

Starting Activities

Start an Activity with an explicit intent

To start a specific `Activity`, 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);`

Start an Activity with implicit intent

To ask Android to find an `Activity` to handle your request, use an implicit `Intent`

1. Create an `Intent`
   - `Intent intent = new Intent(action, uri);`
2. Use the `Intent` to start the `Activity`
   - `startActivity(intent);`

Implicit Intents - Examples

Show a web page

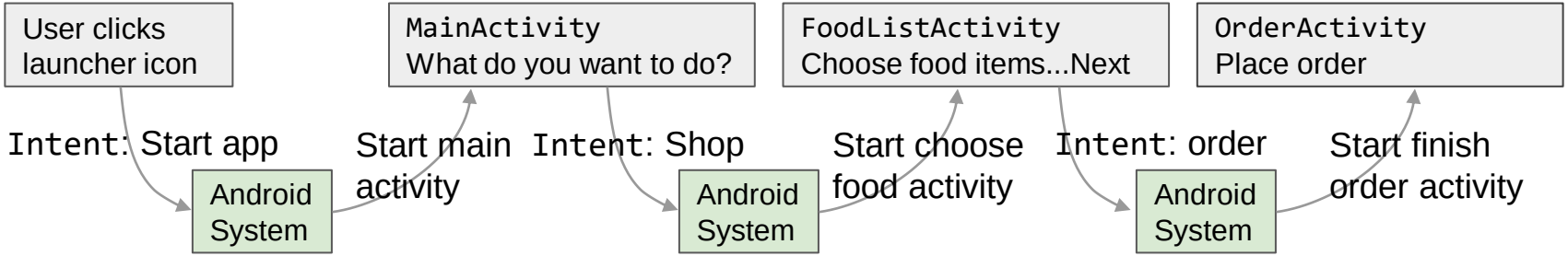
```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

Dial a phone number

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

How Activities Run

- All `Activity` instances are managed by the Android runtime
- Started by an "`Intent`", a message to the Android runtime to run an activity



Sending and Receiving Data

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 and retrieving data

In the first (sending) `Activity`:

1. Create the `Intent` object
2. Put data or extras into that `Intent`
3. Start the new `Activity` with `startActivity()`

In the second (receiving) `Activity`:

1. Get the `Intent` object, the `Activity` was started with
2. Retrieve the data or extras from the `Intent` object

Put information into intent extras

- `putExtra(String name, int value)`
  ⇒ `intent.putExtra("level", 406);`
- `putExtra(String name, String[] value)`
  ⇒ `String[] foodList = {"Rice", "Beans", "Fruit"};`
  `intent.putExtra("food", foodList);`
- `putExtras(bundle);`
  ⇒ if lots of data, first create a bundle and pass the bundle.
- See documentation for all

Sending data to an activity with extras

```
public static final String EXTRA_MESSAGE_KEY =
    "com.example.android.twoactivities.extra.MESSAGE";

Intent intent = new Intent(this,
SecondActivity.class);
String message = "Hello Activity!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

Returning data to the starting activity

1. Use `startActivityForResult()` to start the second `Activity`
2. To return data from the second `Activity`:
   - Create a ***new*** `Intent`
   - Put the response data in the `Intent` using `putExtra()`
   - Set the result to `Activity.RESULT_OK` or `RESULT_CANCELED`, if the user cancelled out
   - call `finish()` to close the `Activity`

1. Implement `onActivityResult()` in first `Activity`

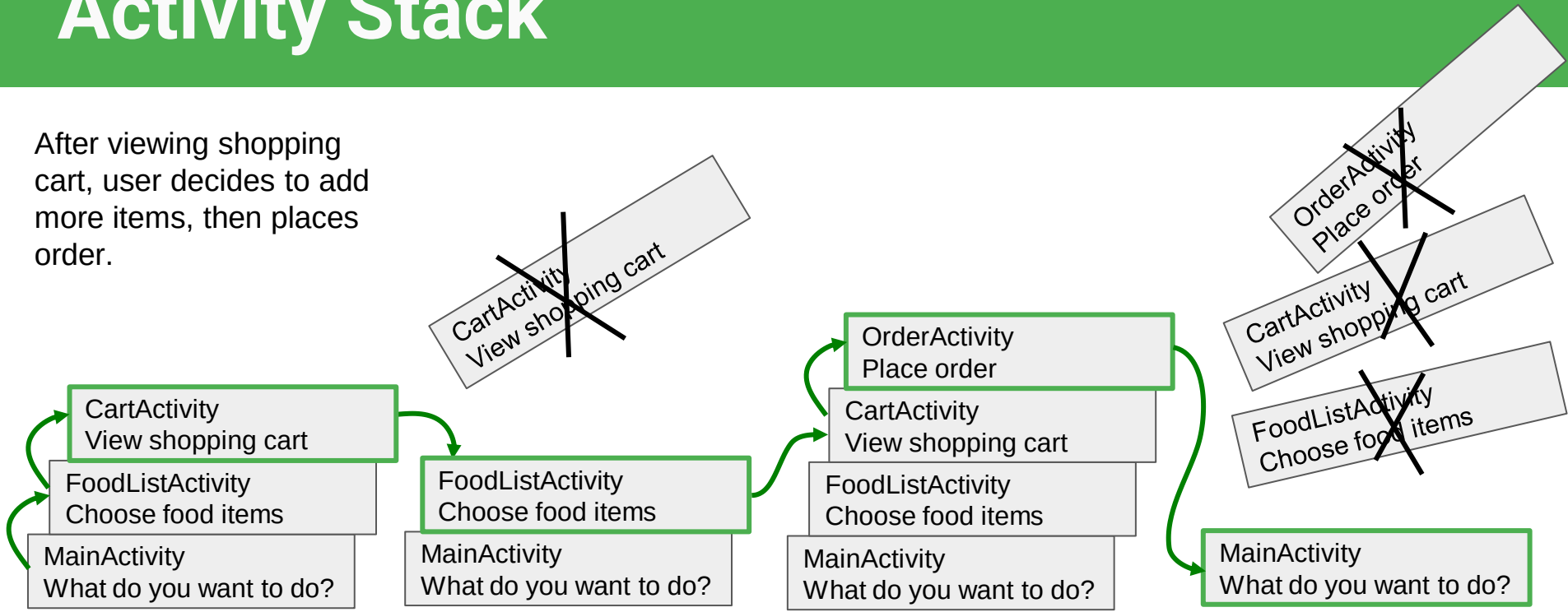
Navigation

Activity stack

- When a new `Activity` is started, the previous `Activity` is stopped and pushed on the `Activity` back stack
- Last-in-first-out-stack—when the current `Activity` ends, or the user presses the Back button, it is popped from the stack and the previous `Activity` resumes

Activity Stack

After viewing shopping cart, user decides to add more items, then places order.



Two forms of navigation

Temporal or back navigation

- provided by the device's Back button
- controlled by the Android system's back stack

Ancestral or up navigation

- provided by the Up button in app's action bar
- controlled by defining parent-child relationships between activities in the Android manifest

Back navigation

- Back stack preserves history of recently viewed screens
- Back stack contains all the `Activity` instances that have been launched by the user in reverse order *for the current task*
- Each task has its own back stack
- Switching between tasks activates that task's back stack

Up navigation

- Goes to parent of current `Activity`
- Define an `Activity` parent in Android manifest
- Set `parentActivityName`

```
<activity
    android:name=".ShowDinnerActivity"
    android:parentActivityName=".MainActivity" >
</activity>
```

Learn more

Learn more

- Android Application Fundamentals
- Starting Another `Activity`
- `Activity` (API Guide)
- `Activity` (API Reference)
- `Intents and Intent Filters` (API Guide)
- `Intent` (API Reference)
- Navigation

What's Next?

- Concept Chapter: 2.1 Activities and Intents
- Practical: 2.1 Activities and intents

END

Android Developer Fundamentals V2

User Interaction

Lesson 4

4.3 Menus and pickers

Contents

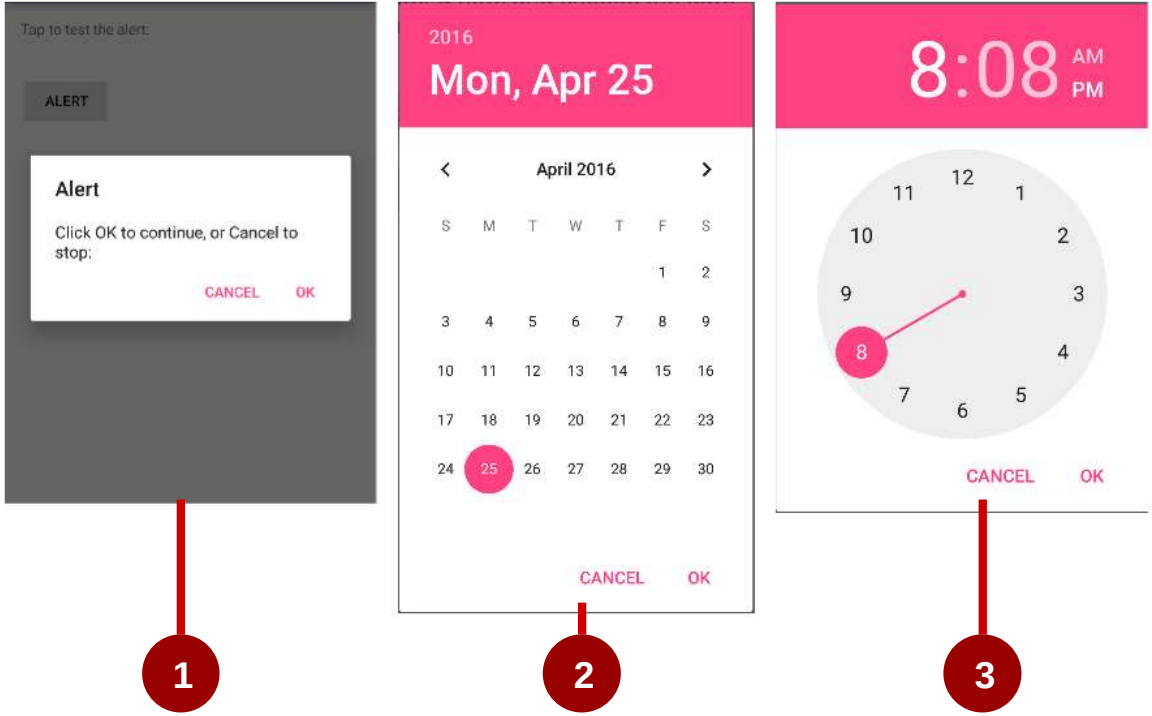
Overview

Types of Menus

1. App bar with options menu
2. Context menu
3. Contextual action bar
4. Popup menu

Dialogs and pickers

1. Alert dialog
2. Date picker
3. Time picker

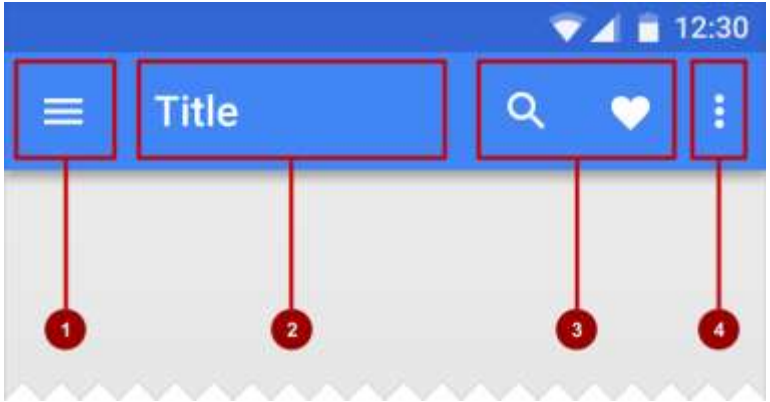


App Bar with Options Menu

What is the App Bar?

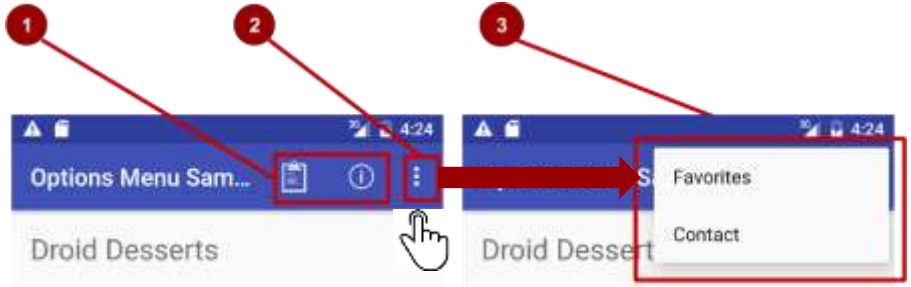
Bar at top of each screen—same for all devices (usually)

1. Nav icon to open navigation drawer
2. Title of current Activity
3. Icons for options menu items
4. Action overflow button for the rest of the options menu



What is the options menu?

- Action icons in the app bar for important items (1)
- Tap the three dots, the "action overflow button" to see the options menu (2)

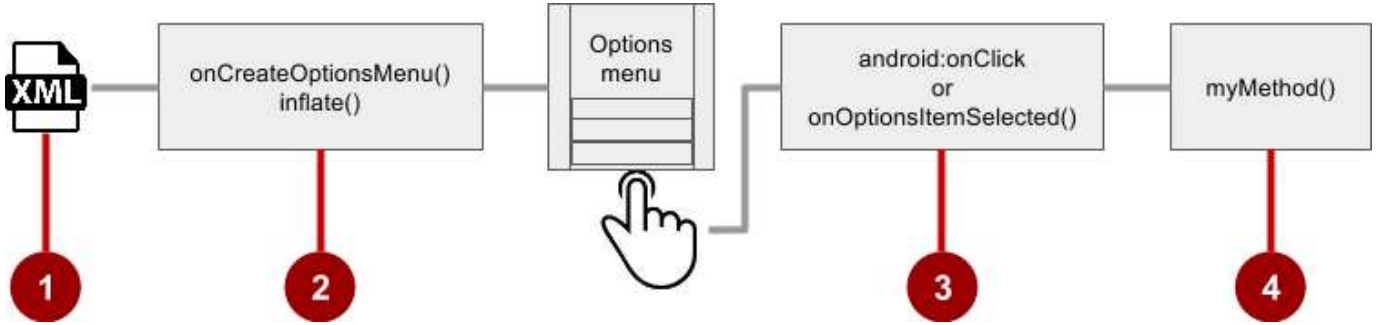


- Appears in the right corner of the app bar (3)
- For navigating to other activities and editing app settings

Adding Options Menu

Steps to implement options menu

1. XML menu resource (`menu_main.xml`)
2. `onCreateOptionsMenu()` to inflate the menu
3. `onClick` attribute or `onOptionsItemSelected()`
4. Method to handle item click



Add icons for menu items

1. Right-click **drawable**
2. Choose **New > Image Asset**
3. Choose **Action Bar and Tab Items**
4. Edit the icon name
5. Click clipart image, and click icon
6. Click **Next**, then **Finish**

Contextual Menus

What are contextual menus?

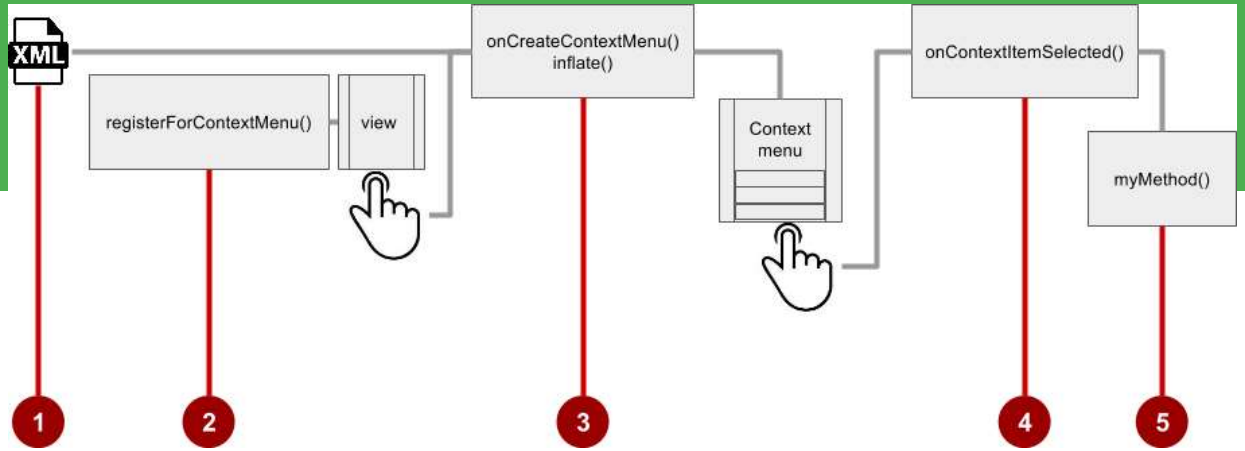
- Allows users to perform action on selected View
- Can be deployed on any View
- Most often used for items in RecyclerView, GridView, or other View collection

Types of contextual menus

- Floating context menu—long-press on a View
  - User can modify View or use it in some fashion
  - User performs action on one View at a time
- Contextual action mode—temporary action bar in place of or underneath app bar
  - Action items affect the selected View element(s)
  - User can perform action on multiple View elements at once

Floating Context Menu

Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Register View using `registerForContextMenu()`
3. Implement `onCreateContextMenu()` in Activity to inflate menu
4. Implement `onContextItemSelected()` to handle menu item clicks
5. Create method to perform action for each context menu item

Contextual Action Bar

What is Action Mode?

- UI mode that lets you replace parts of normal UI interactions temporarily
- For example: Selecting a section of text or long-pressing an item could trigger action mode

Action mode has a lifecycle

- Start it with startActionMode(), for example, in the listener
- ActionMode.Callback interface provides lifecycle methods you override:
  - `onCreateActionMode(ActionMode, Menu)` once on initial creation
  - `onPrepareActionMode(ActionMode, Menu)` after creation and any time ActionMode is invalidated
  - `onActionItemClicked(ActionMode, MenuItem)` any time contextual action button is clicked
  - `onDestroyActionMode(ActionMode)` when action mode is closed

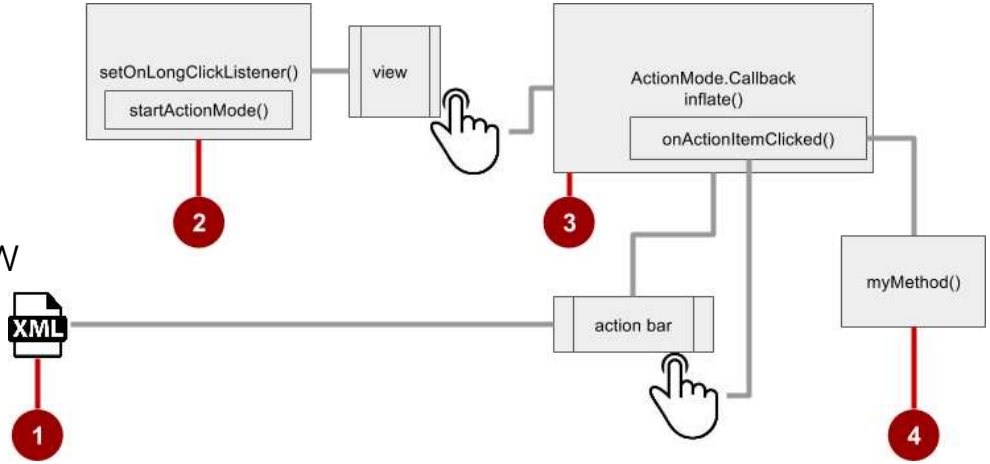
What is a contextual action bar?

Long-press on View shows contextual action bar

1. Contextual action bar with actions
   - **Edit**, **Share**, and **Delete**
   - Done (left arrow icon) on left side
   - Action bar is available until user taps Done
2. View on which long press triggers contextual action bar

Steps for contextual action bar

1. Create XML menu resource file and assign icons for items
2. `setOnLongClickListener()` on View that triggers contextual action bar and call `startActionMode()` to handle click
3. Implement `ActionMode.Callback` interface to handle ActionMode lifecycle; include action for menu item click in `onActionItemClicked()` callback
4. Create method to perform action for each context menu item



Use setOnLongClickListener

```
private ActionMode mActionMode;
```

In onCreate():

```
View view = findViewById(article);
view.setOnLongClickListener(new View.OnLongClickListener() {
   public boolean onLongClick(View view) {
      if (mActionMode != null) return false;
      mActionMode =
            MainActivity.this.startActionMode(mActionModeCallback);
      view.setSelected(true);
      return true;
   }
});
```

Implement mActionModeCallback

```
public ActionMode.Callback mActionModeCallback =
   new ActionMode.Callback() {
    // Implement action mode callbacks here.
};
```

Implement onCreateActionMode

```
@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
    return true;
}
```

Implement onPrepareActionMode

- Called each time action mode is shown
- Always called after `onCreateActionMode`, but may be called multiple times if action mode is invalidated

```
@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
   return false; // Return false if nothing is done.
}
```

Implement onActionItemClicked

- Called when users selects an action
- Handle clicks in this method

```
@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
   switch (item.getItemId()) {
       case R.id.action_share:
         // Perform action for the Share menu item.
         mode.finish(); // Action picked, so close the action bar.
         return true;
       default:
         return false;
   }
}
```

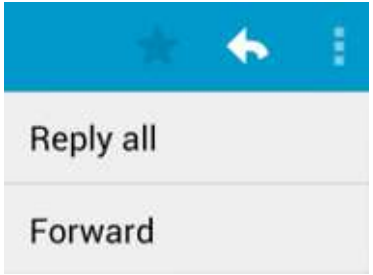
Implement onDestroyActionMode

- Called when user exits the action mode

```
@Override
public void onDestroyActionMode(ActionMode mode) {
    mActionMode = null;
}
```

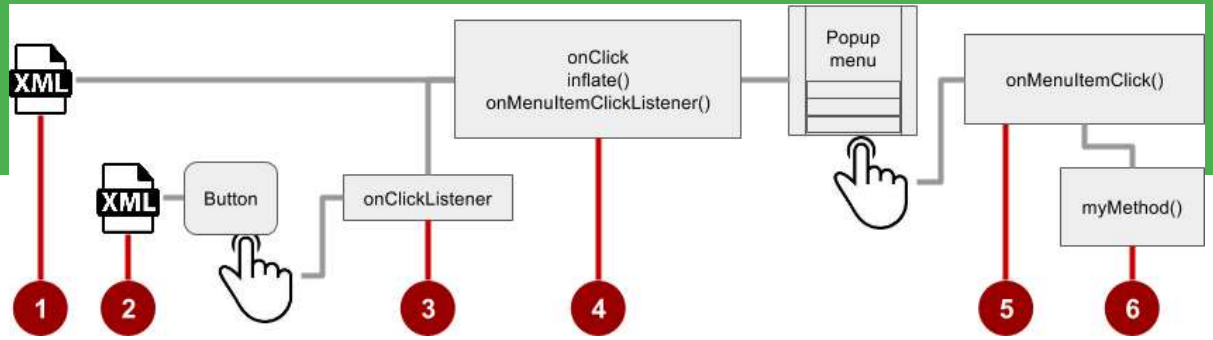
Popup Menu

What is a popup menu?



- Vertical list of items anchored to a view
- Typically anchored to a visible icon
- Actions should not directly affect view content
  - Options menu overflow icon that opens options menu
  - In email app, **Reply All** and **Forward** relate to email message but don't affect or act on message

Steps



1. Create XML menu resource file and assign appearance and position attributes
2. Add `ImageButton` for the popup menu icon in the XML activity layout file
3. Assign `onClickListener` to `ImageButton`
4. Override `onClick()` to inflate the popup and register it with `onMenuItemClickListener()`
5. Implement `onMenuItemClick()`
6. Create a method to perform an action for each popup menu item

Add ImageButton

```
<ImageButton
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/button_popup"
 android:src="@drawable/@drawable/ic_action_popup"/>
```

Assign onClickListener to button

```
private ImageButton mButton =
    (ImageButton) findViewById(R.id.button_popup);
```

In onCreate():

```
mButton.setOnClickListener(new View.OnClickListener() {
    // define onClick
});
```

Implement onClick

```
@Override
public void onClick(View v) {
    PopupMenu popup = new PopupMenu(MainActivity.this, mButton);
    popup.getMenuInflater().inflate(
        R.menu.menu_popup, popup.getMenu());
    popup.setOnMenuItemClickListener(
        new PopupMenu.OnMenuItemClickListener() {
            // implement click listener.
        });
    popup.show();
}
```

Implement onMenuItemClick

```
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.option_forward:
            // Implement code for Forward button.
            return true;
        default:
            return false;
    }
}
```

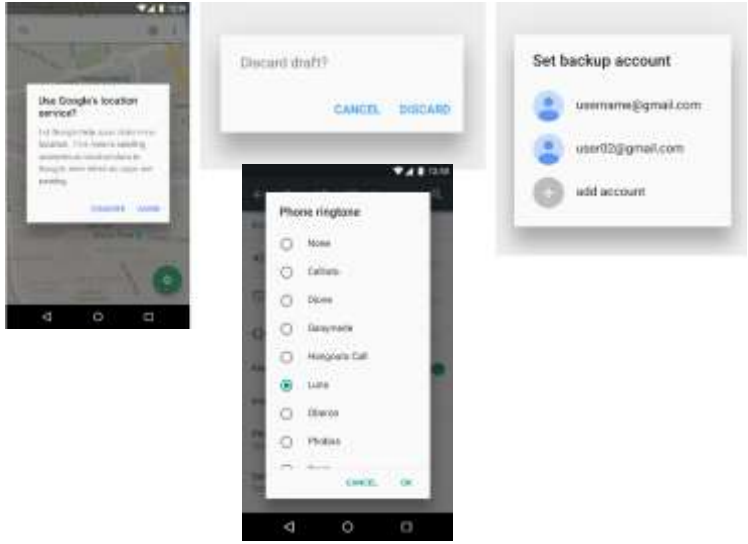
Dialogs

Dialogs

- Dialog appears on top, interrupting flow of Activity
- Requires user action to dismiss

TimePickerDialog

DatePickerDialog

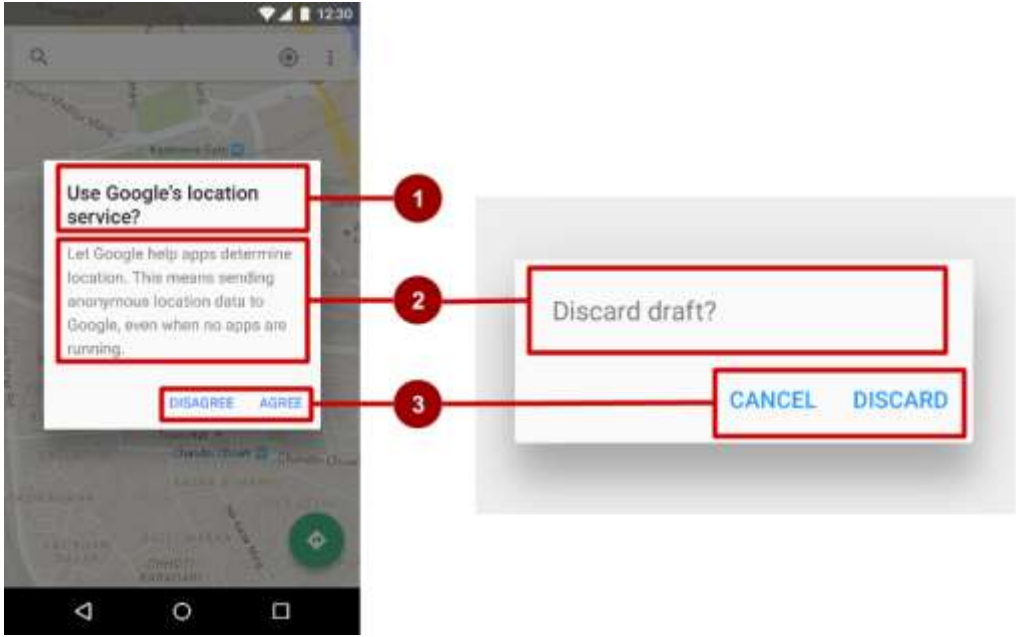


AlertDialog

AlertDialog

AlertDialog can show:

1. Title (optional)
2. Content area
3. Action buttons



Build the AlertDialog

Use **`AlertDialog.Builder`** to build alert dialog and set attributes:

```
public void onClickShowAlert(View view) {
    AlertDialog.Builder alertDialog = new
                    AlertDialog.Builder(MainActivity.this);
    alertDialog.setTitle("Connect to Provider");
    alertDialog.setMessage(R.string.alert_message);
    // ... Code to set buttons goes here.
```

Set the button actions

- `alertDialog.setPositiveButton()`
- `alertDialog.setNeutralButton()`
- `alertDialog.setNegativeButton()`

alertDialog code example

```
alertDialog.setPositiveButton(
                    "OK", newDialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
        // User clicked OK button.
    }
});
```

Same pattern for setNegativeButton() and setNeutralButton()

Pickers

Pickers

- DatePickerDialog
- TimePickerDialog

Pickers use fragments

- Use DialogFragment to show a picker
- DialogFragment is a window that floats on top of Activity window

Introduction to fragments

- A Fragment is like a mini-Activity within an Activity
  - Manages its own own lifecycle
  - Receives its own input events
- Can be added or removed while parent Activity is running
- Multiple fragments can be combined in a single Activity
- Can be reused in more than one Activity

Creating a date picker dialog

1. Add a blank `Fragment` that extends `DialogFragment` and implements `DatePickerDialog.OnDateSetListener`
2. In `onCreateDialog()` initialize the date and return the dialog
3. In `onDateSet()` handle the date
4. In `Activity` show the picker and add method to use date

Creating a time picker dialog

1. Add a blank `Fragment` that extends `DialogFragment` and implements `TimePickerDialog.OnTimeSetListener`
2. In `onCreateDialog()` initialize the time and return the dialog
3. In `onTimeSet()` handle the time
4. In Activity, show the picker and add method to use time

Learn more

- Adding the App Bar
- Menus
- Menu Resource
- Fragments
- Dialogs
- Pickers
- Drawable Resources

What's Next?

- Concept Chapter: 4.3 Menus and pickers
- Practical: 4.3 Menus and pickers

END