Recording in Progress

This class is being recorded

Please turn off your video and/or video if you do not wish to be recorded

CMSC436: Programming Handheld Systems

Application Fundamentals

Application Components

Activity

Service

BroadcastReceiver

ContentProvider

Applications

Apps are made from components

Android instantiates and runs them as needed

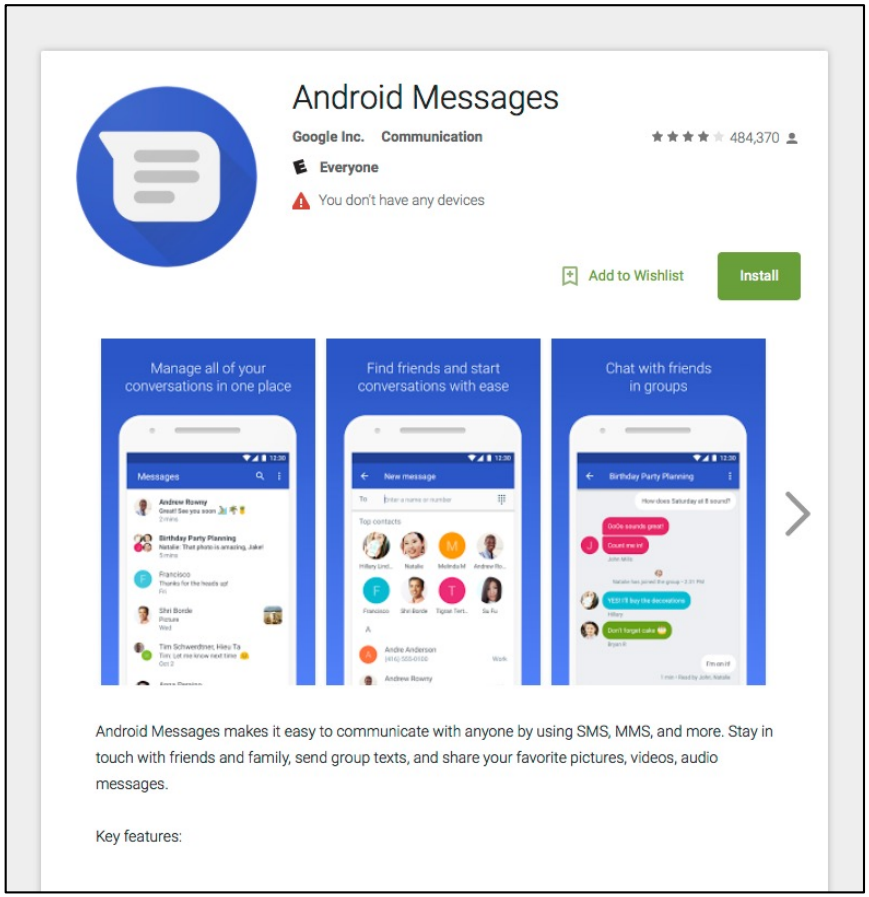
Each component has its own purpose and APIs

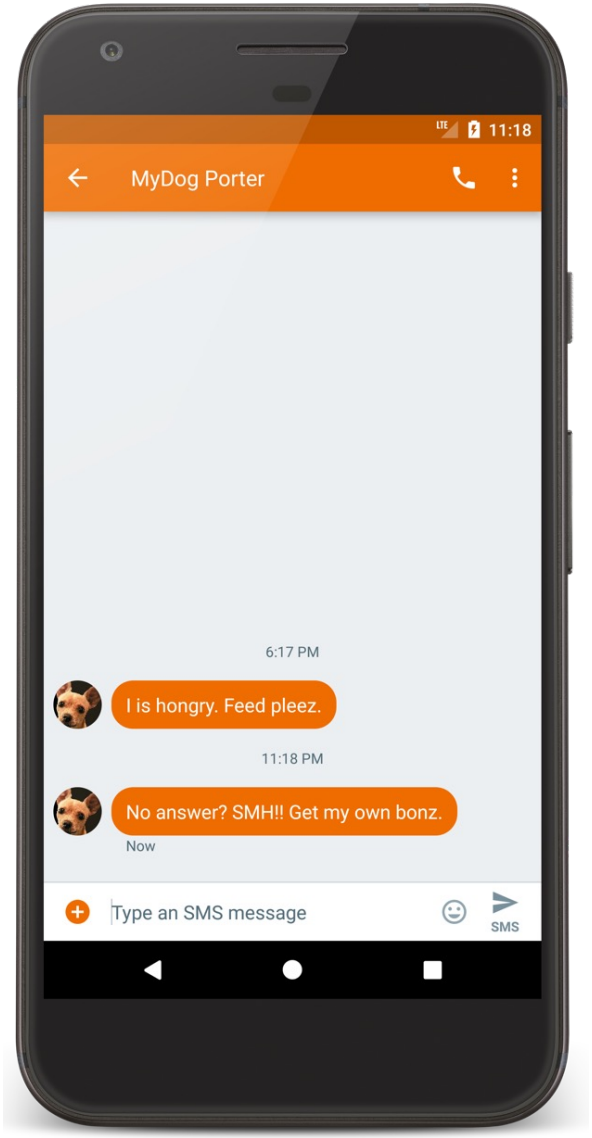
Apps can have multiple “entry points”

Activity

- Primary class for user interaction
- Conceptually implements a single, focused task that the user can do
  - Implementation changes slightly with introduction of Fragments (discussed later in the course)

Example App
Android Messages




11:18
MyDog Porter
6:17 PM
I is hongry. Feed pleez.
11:18 PM
No answer? SMH!! Get my own bonz.
Now
Type an SMS message
SMS

ConversationActivity.java

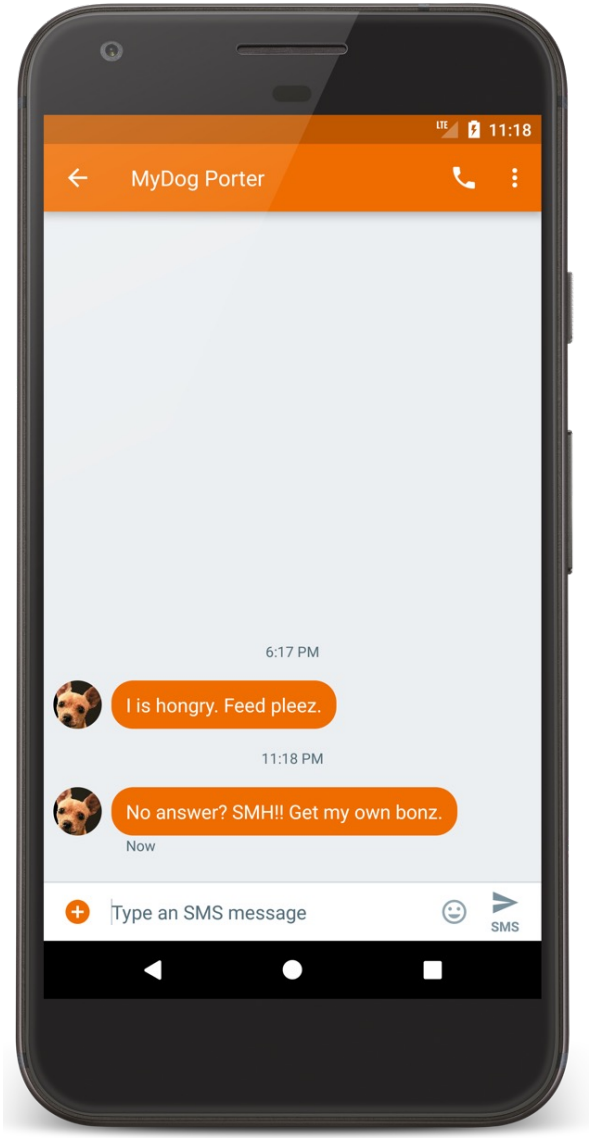
```
package com.android.messaging.ui.conversation;

…
public class ConversationActivity extends BugleActionBarActivity
        implements ContactPickerFragmentHost,
    ConversationFragmentHost, ConversationActivityUiStateHost {

  …
```

Android source code available at: https://source.android.com

Service

Runs in the background

- to perform long-running operations
- to support interaction with remote processes


11:18
MyDog Porter
6:17 PM
I is hongry. Feed pleez.
11:18 PM
No answer? SMH!! Get my own bonz.
Now
Type an SMS message
SMS

MmsService.java

```
package com.android.mms.service;

...
/**
 * System service to process MMS API requests
 */
public class MmsService extends Service implements
                                                MmsRequest.RequestManager {

...
```

BroadcastReceiver

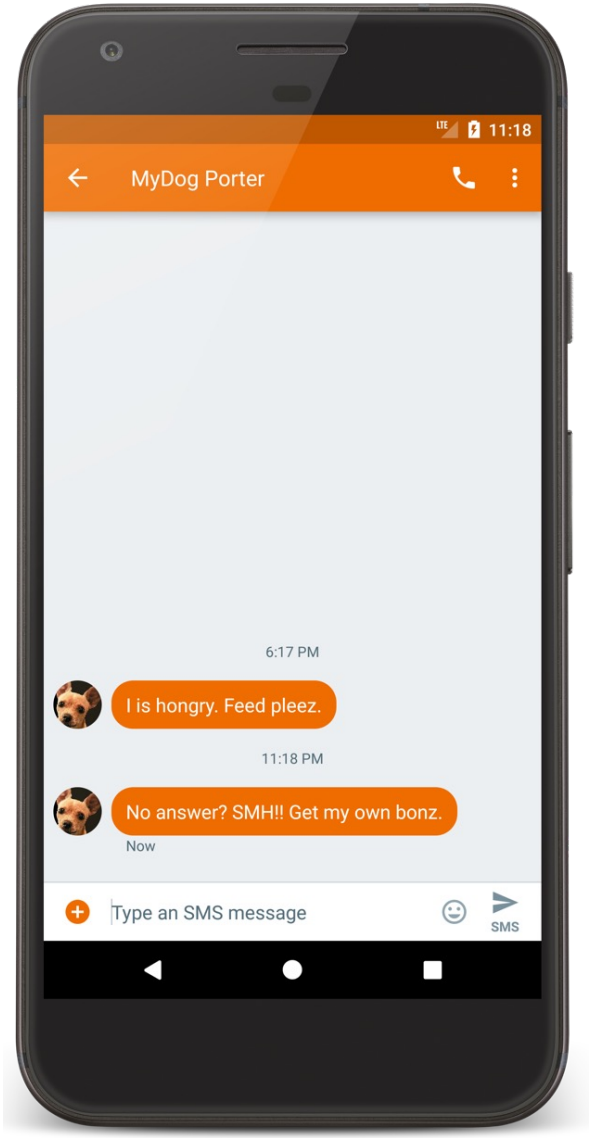
Component that listens for and responds to events

Acts as the subscriber in publish/subscribe pattern

BroadcastReceiver

Events are represented by an Intent and then broadcast by the platform

BroadcastReceivers can receive and respond to to broadcast events

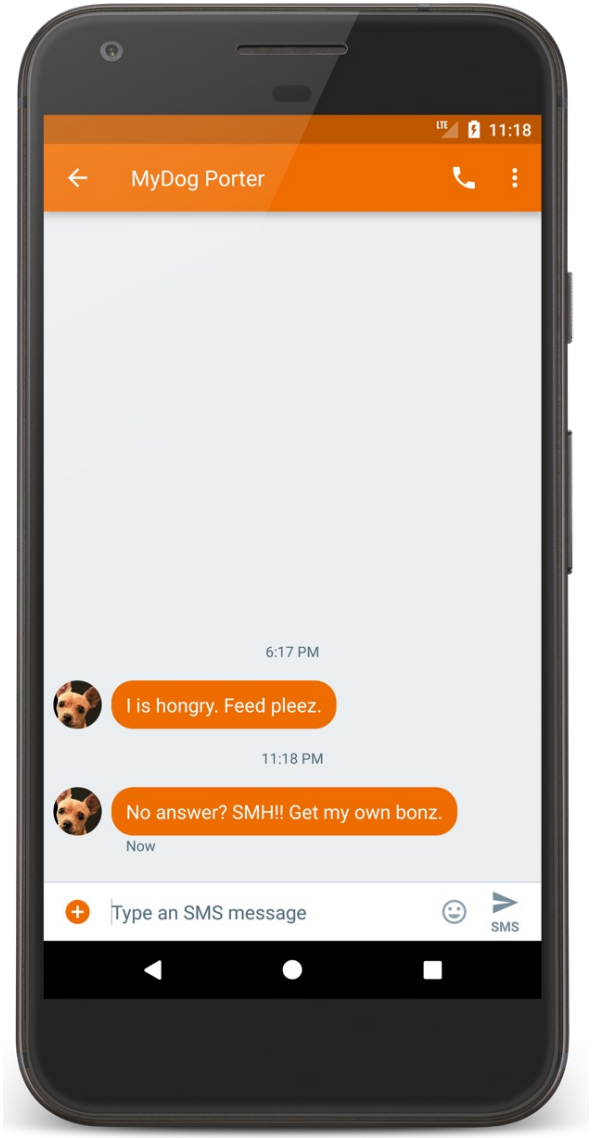

11:18
MyDog Porter
6:17 PM
I is hongry. Feed pleez.
11:18 PM
No answer? SMH!! Get my own bonz.
Now
Type an SMS message
SMS

SmsDeliverReceiver.java

```
package com.android.messaging.receiver;
...
/**
 * Class that receives incoming SMS messages on KLP+ Devices.
 */
public final class SmsDeliverReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(final Context context, final Intent intent) {
        SmsReceiver.deliverSmsIntent(context, intent);
    }
}
```

Content Providers

- Store & share data across applications
  - Uses database-style interface
  - Handles interprocess communication


11:18
MyDog Porter
6:17 PM
I is hongry. Feed pleez.
11:18 PM
No answer? SMH!! Get my own bonz.
Now
Type an SMS message
SMS

SuggestionsProvider.java

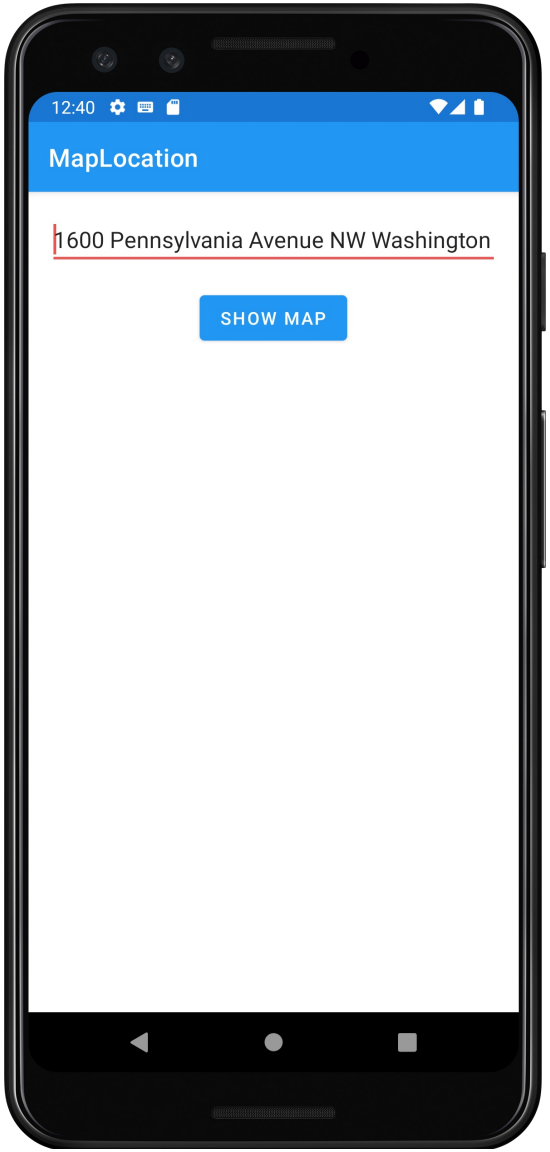
```
package com.android.mms;
...
/**
 * Suggestions provider for mms.
* Queries the "words" table to provide possible word suggestions.
*/

public class SuggestionsProvider extends android.content.ContentProvider {
    ...
```

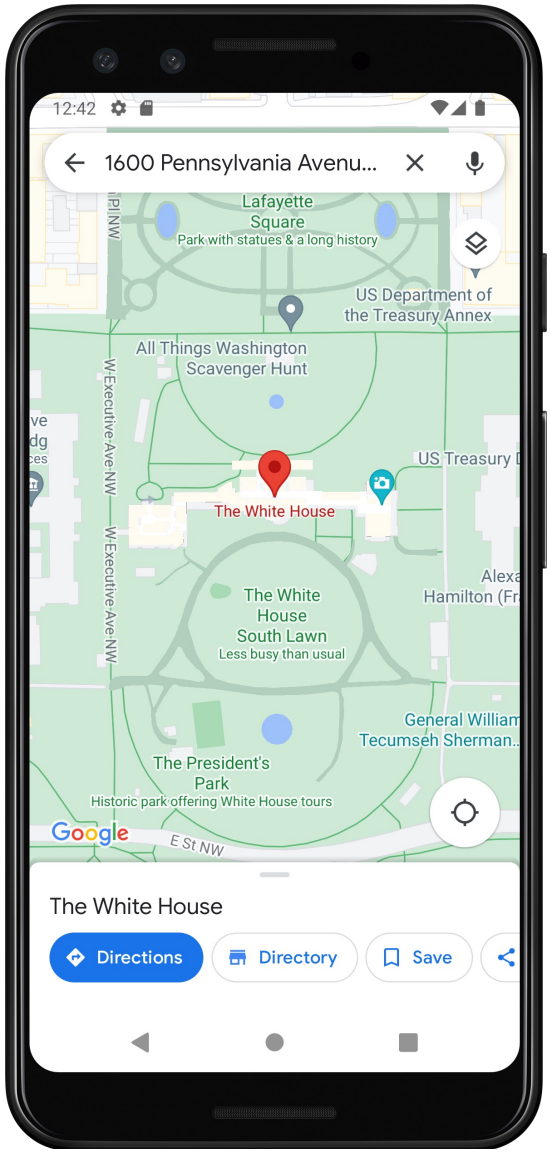
MapLocation

User enters an address

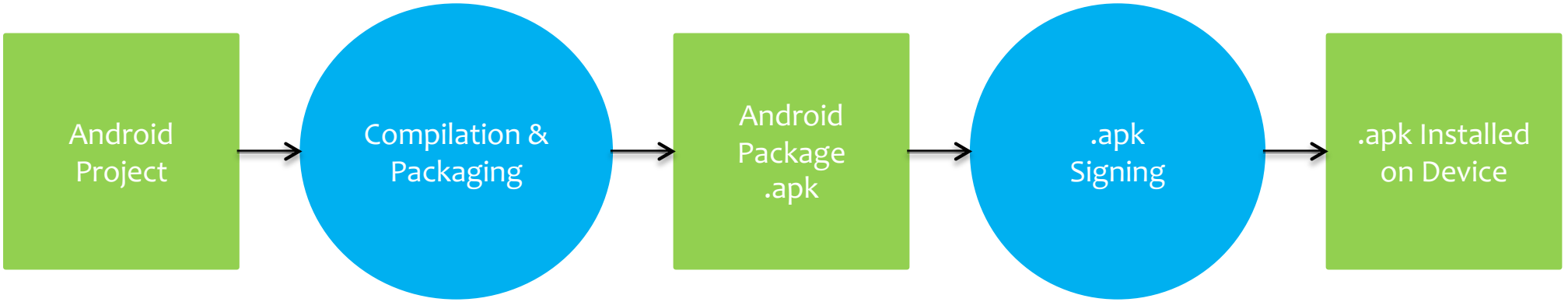
App displays a map of area around the address

MapLocation




12:42
1600 Pennsylvania Avenu...
Lafayette Square
Park with statues & a long history
US Department of the Treasury Annex
All Things Washington Scavenger Hunt
W Executive Ave NW
The White House
US Treasury
The White House South Lawn
Less busy than usual
The President's Park
Historic park offering White House tours
Google
E St NW
The White House
Directions
Directory
Save

Simplified App Development Workflow



Creating an Android App

Define resources

Implement application classes

Package application

Install & run application

1. Defining Resources

Resources are non-source code entities

Many different resource types, e.g.,

- Layout, strings, images, menus, & animations

Allows apps to be customized for different devices and users

See: https://developer.android.com/guide/topics/resources/overview.html

Strings

Types: String, String Array, Plurals

Strings

Types: String, String Array, Plurals

Typically stored in res/values/*.xml

Specified in XML, e.g.,

```
<string name="hello">Hello World!</string>
```

Can include formatting and styling codes

Strings

Accessed by other resources as:

@string/string_name

Accessed in Kotlin as:

R.string.string_name

MapLocation’s Strings Files

values/strings.xml

```
<resources>
    <string name="show_map_string">Show Map</string>
    <string name="location_string">Enter Location</string>
</resources>
```

values-it/strings.xml

```
<resources>
    <string name="show_map_string">Mostra la mappa</string>
    <string name="location_string">Digita l\'indirizzo</string>
</resources>
```

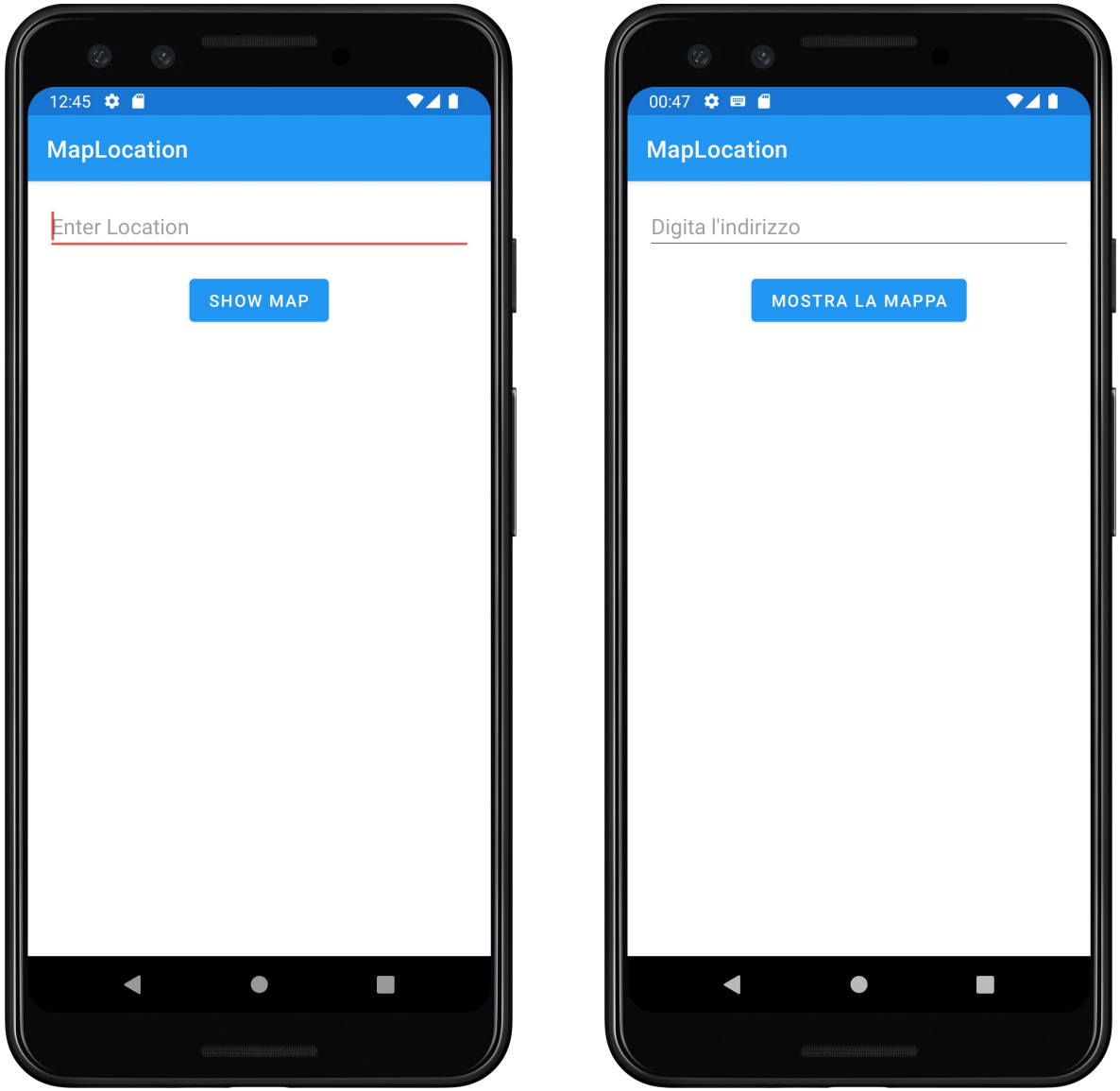
Customized Strings at Runtime

If your default language is Italian, @string/location_string is

> “Digita l’indirizzo”

Otherwise, it’s,

> “Enter Location”

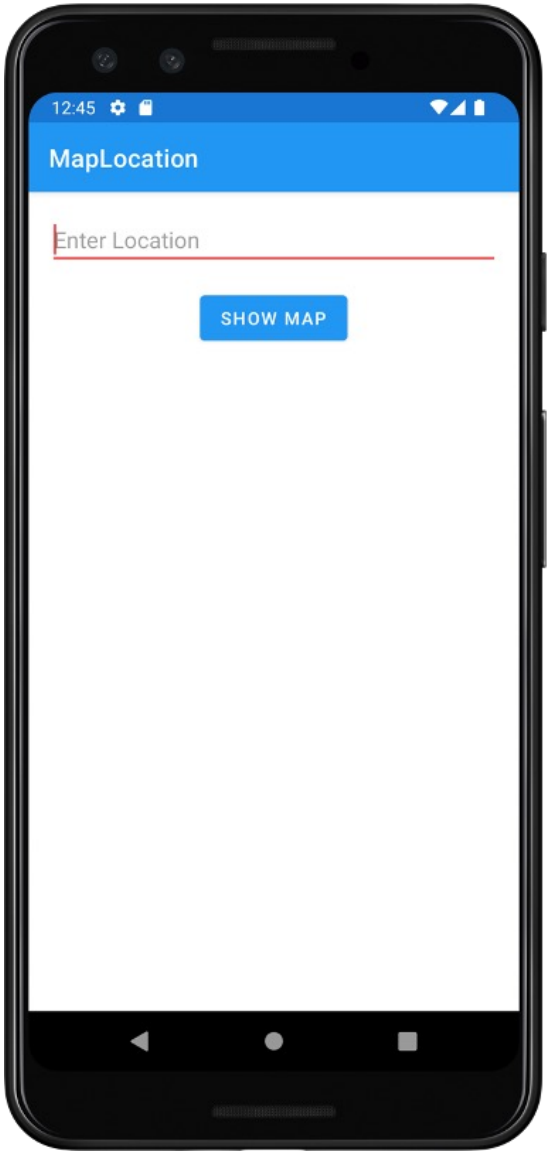

12:45
MapLocation
Enter Location
SHOW MAP
00:47
MapLocation
Digita l'indirizzo
MOSTRA LA MAPPA

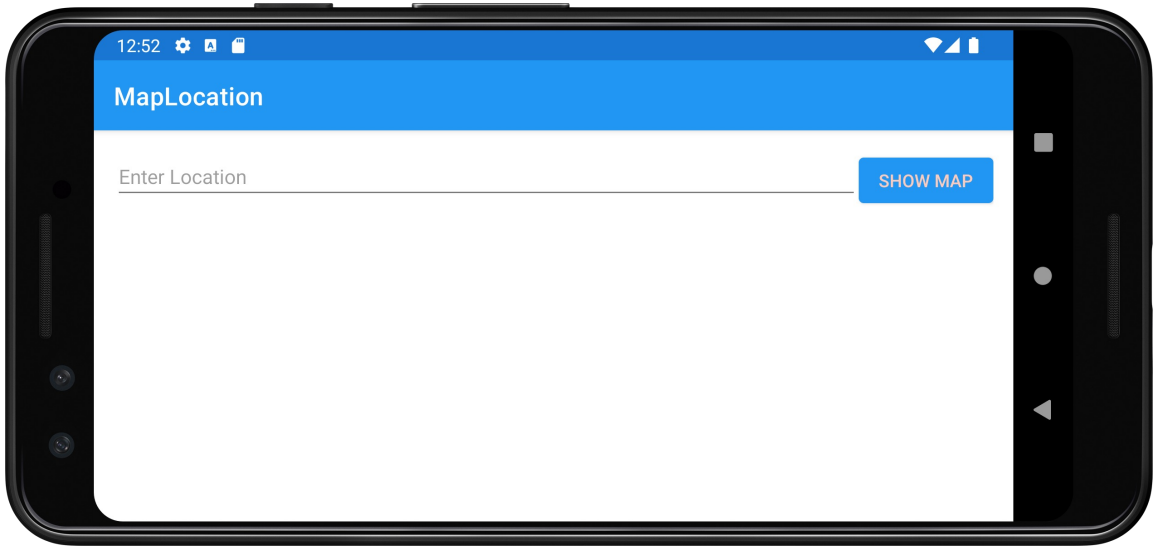
User Interface Layout

- UI layout specified in XML files
  - Some tools allow visual layout
- XML files typically stored in res/layout/*.xml
- Accessed in Kotlin as R.layout. layout_name
- Accessed by other resources as: @layout/layout_name

Using Multiple Layout Files

Can specify different layout files based on your device’s orientation, screen size, etc.


12:45
MapLocation
Enter Location
SHOW MAP


12:52
MapLocation
Enter Location
SHOW MAP

R Class

At compilation time, resources are used to generate the R class

App code uses the R class to access resources

R class is generated directly into bytecode

R.Java (Simulated Example)

```
package course.examples.maplocation;

public final class R {
  public static final class color {
    public static final int accent=0x7f010000;
    public static final int edit_text=0x7f010001;
    public static final int primary=0x7f010002;
    public static final int primary_dark=0x7f010003;
    public static final int primary_light=0x7f010004;
    public static final int primary_text=0x7f010005;
    public static final int secondary_text=0x7f010006;
  }
```

R.Java (Simulated Example)

```
public static final class dimen {
    public static final int activity_margin=0x7f020000;
  }
  public static final class id {
    public static final int location=0x7f030000;
    public static final int mapButton=0x7f030001;
  }
  public static final class layout {
    public static final int main=0x7f040000;
  }
  public static final class mipmap {
    public static final int ic_launcher=0x7f050000;
  }
```

R.Java (Simulated Example)

```
  public static final class string {
    public static final int location_string=0x7f060000;
    public static final int show_map_string=0x7f060001;
  }
  public static final class style {
    public static final int MaterialTheme=0x7f070000;
  }
}
```

2. Implement Classes

Usually involves at least one Activity

Activity initialization code usually in onCreate()

2. Implement Classes

Typical onCreate() workflow

- Restore saved state, if necessary
- Set content view
- Initialize UI elements
- Link UI elements to code actions

Notes on View Binding

View binding allows code to easily interact with views.

If enabled, view binding generates a *binding class* for each XML layout file

Binding class contains direct references to all views that have an ID in the corresponding layout

view binding replaces findViewById()

Review MapLocation.kt

Open MapLocation.kt

Review with at least one other student

Which of the typical workflow steps are represented in that code?

Where are those steps implemented in the code?

3. Package Application

System packages application components & resources into a .apk file

Developers specify required application information in a file called AndroidManifest.xml

AndroidManifest.xml

Information includes:

Application name

Application components

Other

- Required permissions
- Application features
- etc.

4. Install & Run

From IDE run app in the emulator or device

From command line

- Enable USB Debugging on the device
  - See: https://developer.android.com/studio/debug/dev-options.html
- %adb install <path_to_apk>

Next

The Activity Class

Example Applications

MapLocation