Activities and Intents

Objectives

1. Start an Activity in your .apk
2. Finish an Activity
3. Pass arguments to an Activity
4. Get Activity results
5. Start a system Activity

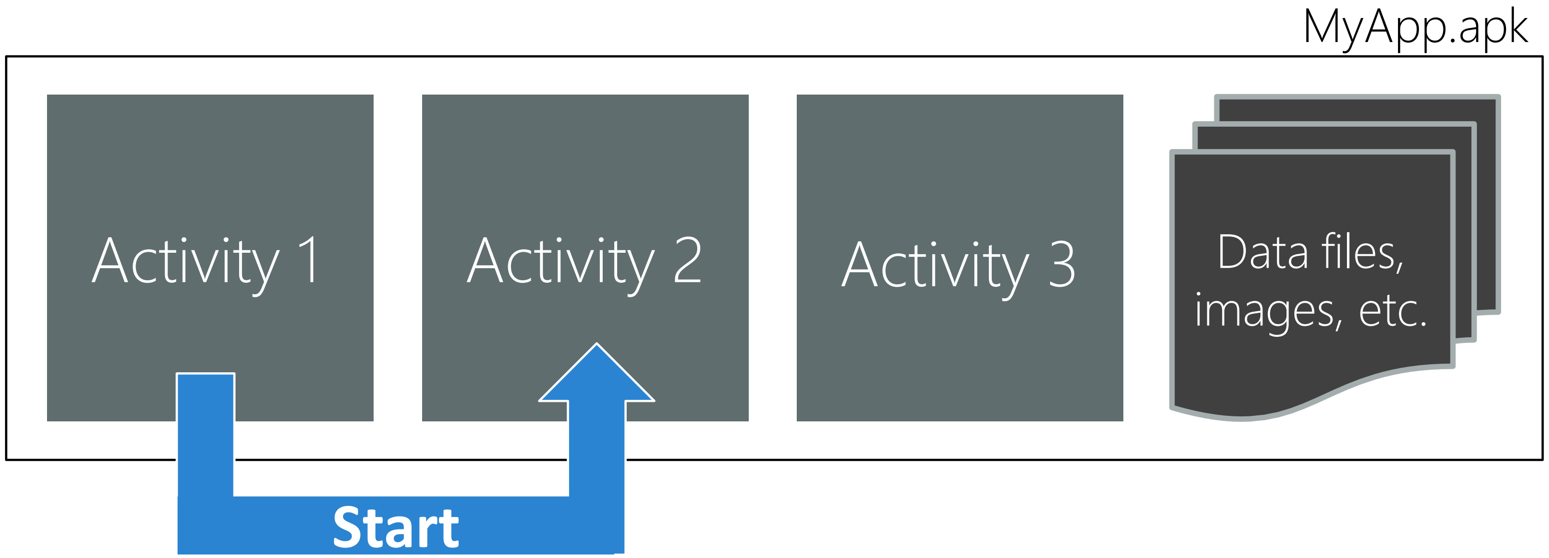
Start an Activity in your .apk

Tasks

1. Create an explicit Intent
2. Start an Activity

Motivation

- An Android app is a collection of collaborating Activities; it is common for one Activity to start another Activity from the same .apk



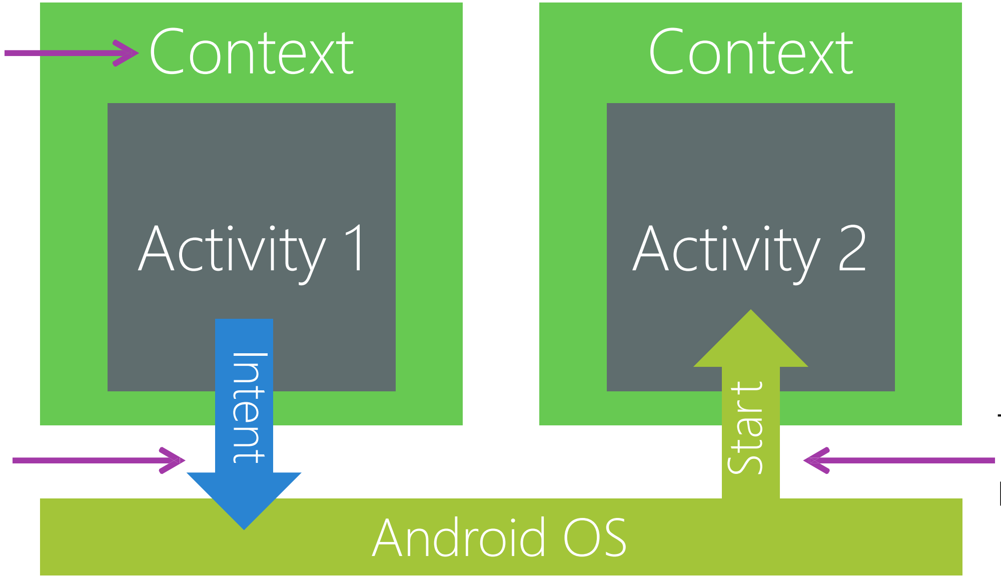
Group Exercise

Explore the completed lab exercise

Activity-start overview

- You need to use a few different Android types to start an Activity

The request goes through a **`Context`**

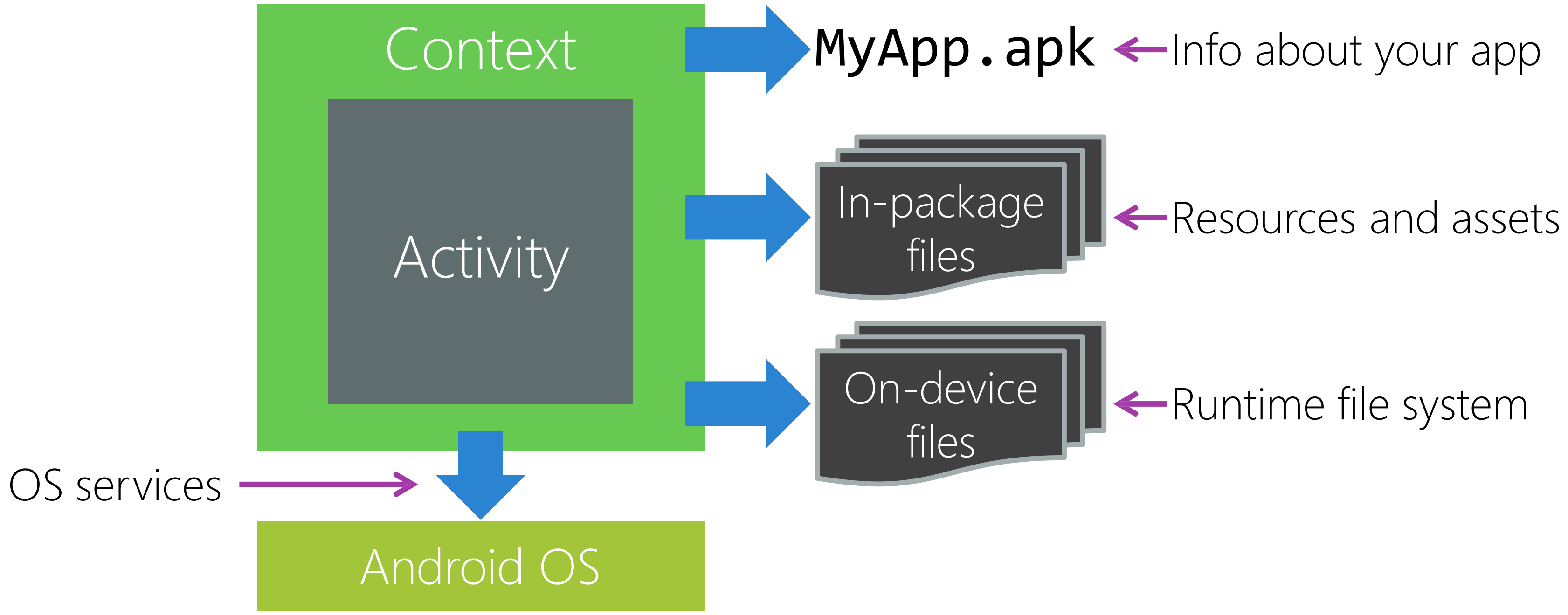


Request packaged inside an **`Intent`**

The OS starts the new Activity

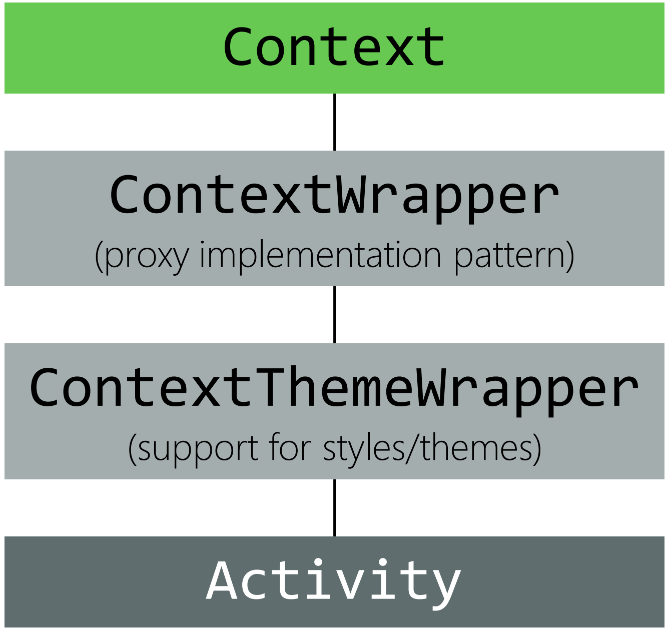
What is a Context?

- *Context* is an access point to the Android environment running your app



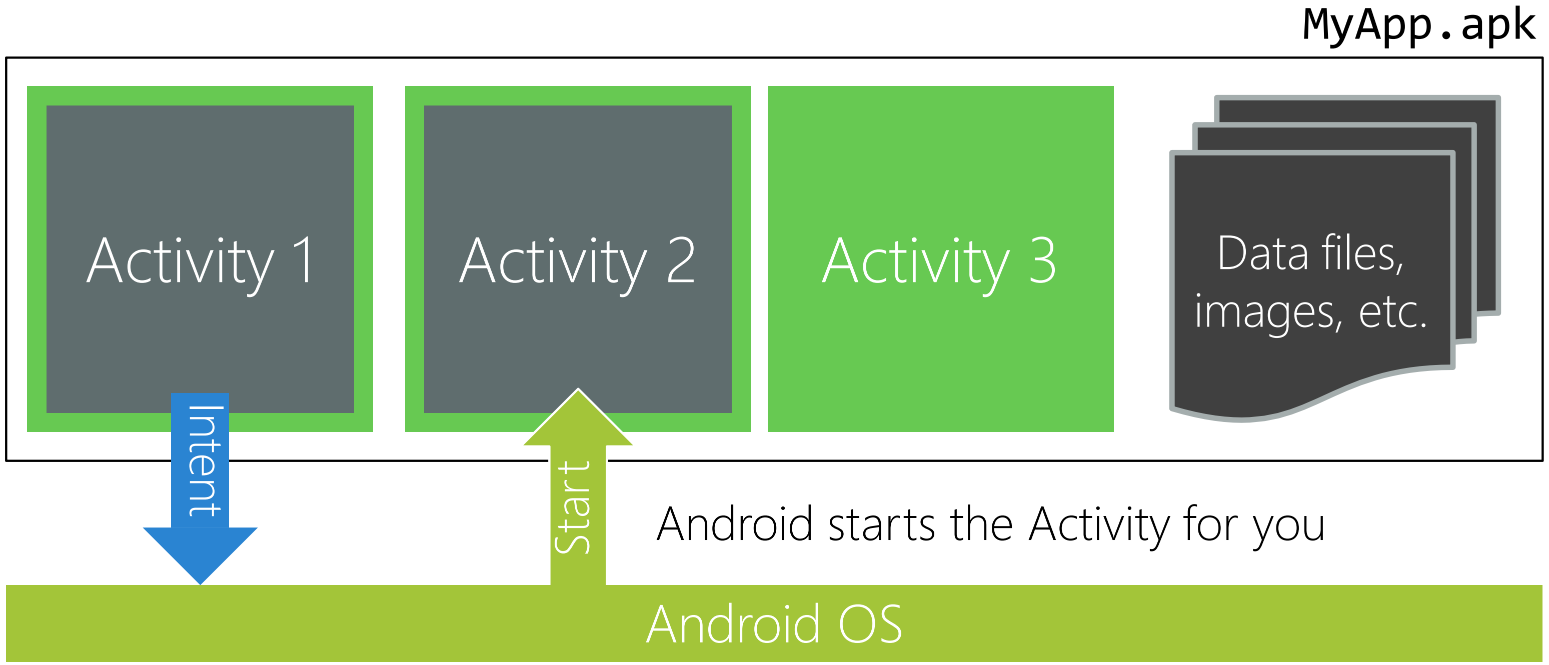
Activity is-a Context

- The **`Activity`** class inherits from **`Context`**
- This ensures each Activity has access to the environment for loading resources and interacting with Android



What is an Intent?

❖ An *Intent* is a request you send to Android to start a new Activity



What is an explicit Intent?

❖ An *explicit Intent* is an Intent that exactly identifies the Activity to start

```
public class Intent : ...
{
  public Intent(Context packageContext, Type type) { ... }
  ...
}
```

This must be a **`Context`** associated with the .apk containing the target Activity (use your current Activity when they are from the same .apk)

`Type` object uniquely identifies the target Activity

Start methods

- Context provides the core methods for starting Activities

Start

Convenience method

```
public abstract class Context : ...
{ ...
  public abstract void StartActivity(Intent intent);

  public void StartActivity(Type type);
}
```

`Context` and **`Activity`** provide other methods to start an Activity; however, the ones shown here are among the most common.

How to start an Activity

❖ To start a new Activity, create an Intent and pass it to **`StartActivity`**

Common to start in response to a user action

Start

```
public class Activity1 : Activity
{ ...
  void OnClick(object sender, EventArgs e)
  {
    var intent = new Intent(this, typeof(Activity2));

    base.StartActivity(intent);
  }
}
```

Individual Exercise

Start an Activity in your .apk

Summary

1. Create an explicit Intent
2. Start an Activity

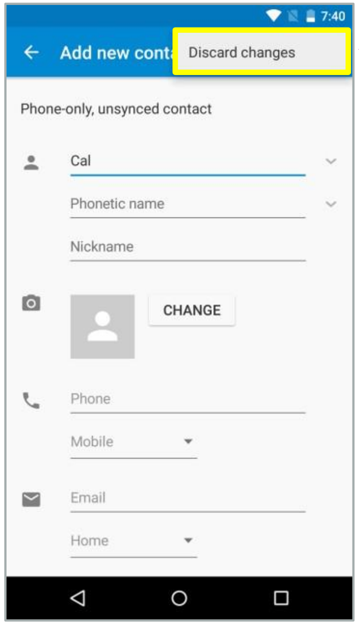
Finish an Activity

Tasks

1. Understand Stack Navigation
2. See the behavior of the Back-button
3. Programmatically finish an Activity

Motivation

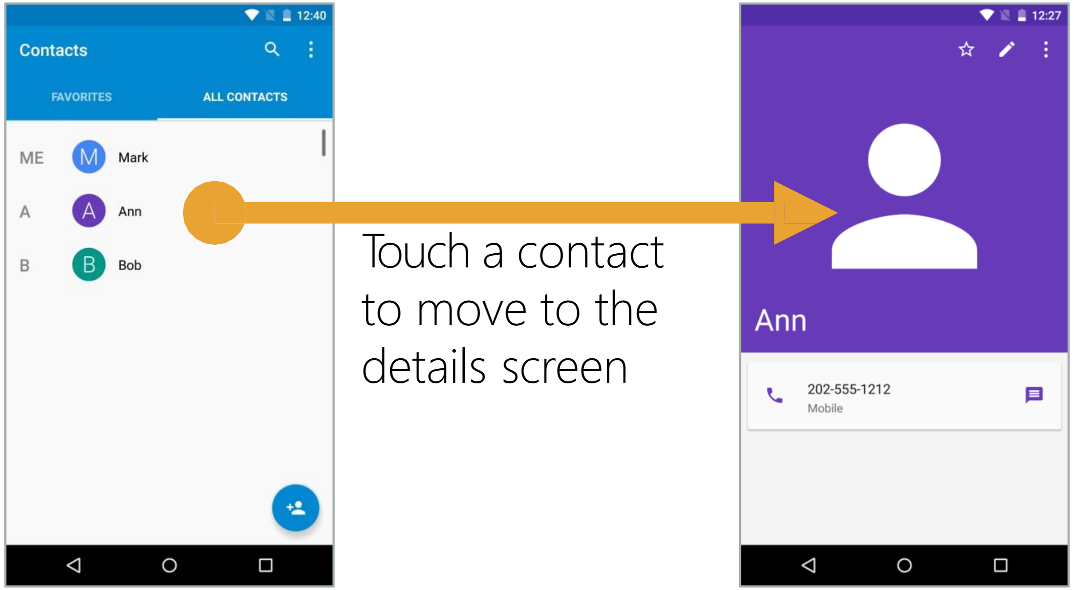
❖ You need to know how to programmatically finish an Activity to implement functionality like "cancel"



The "Add new contact" Activity has a cancel button

What is navigation?

- *Navigation* describes the paths you create in your app to let the user switch between your various Activities

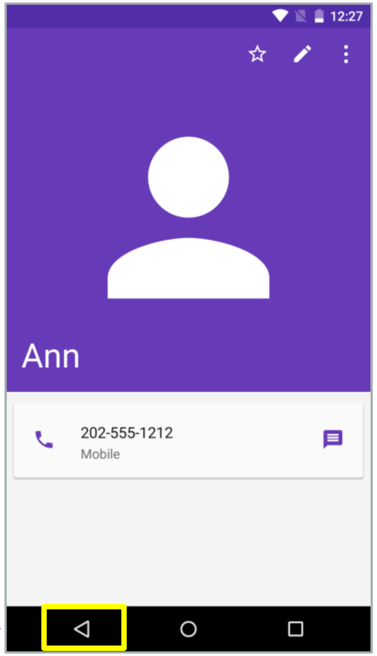


Touch a contact to move to the details screen

What is the Back button?

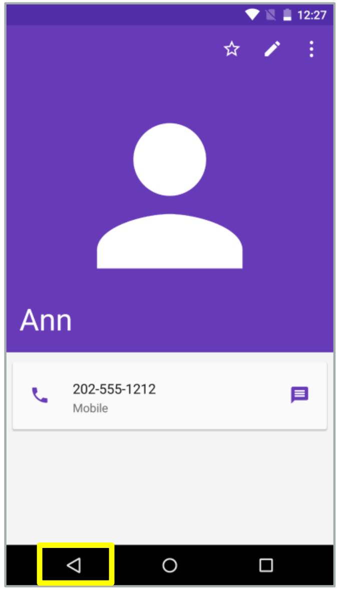
- Android devices have a *Back Button* that returns the user to the previous Activity

The Contacts app lets users move from the All Contacts screen to view an individual contact and then back

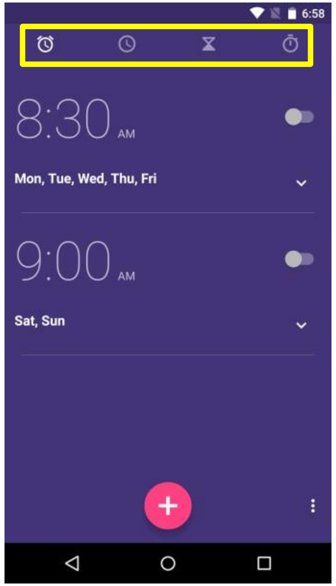


Navigation patterns

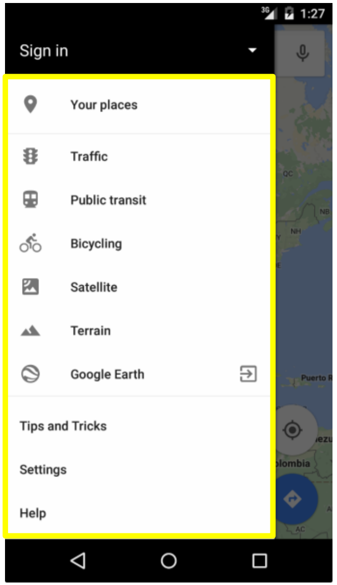
- Android apps use several common navigation patterns



Stack



Tab

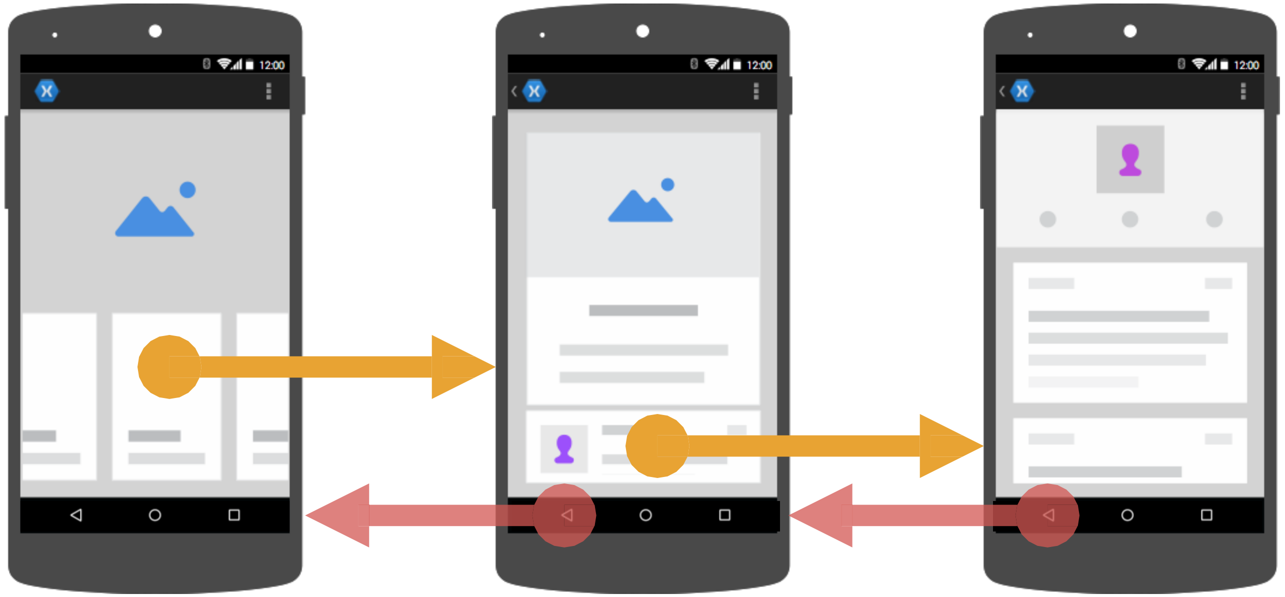


Drawer

This course discusses stack navigation; our navigation course covers other patterns.

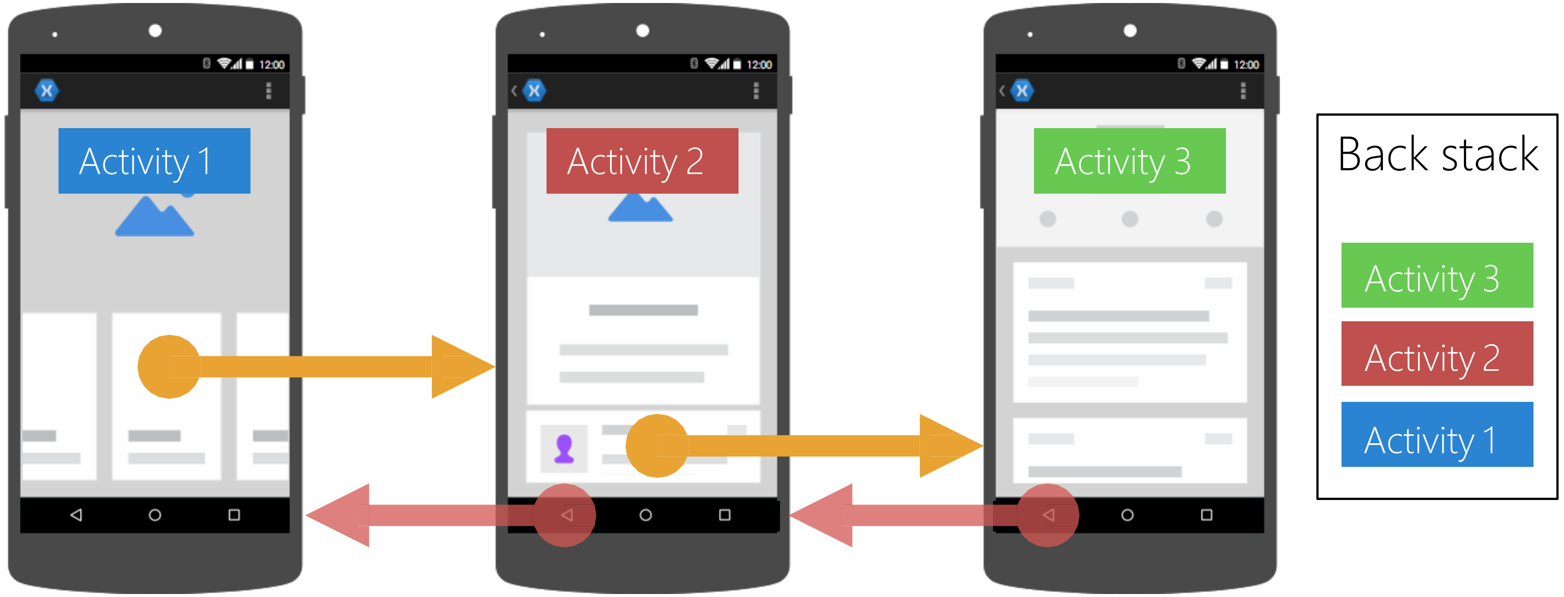
What is stack navigation?

- *Stack navigation* records the sequence of Activities in a stack to enable the user to return from any Activity to the one that started it



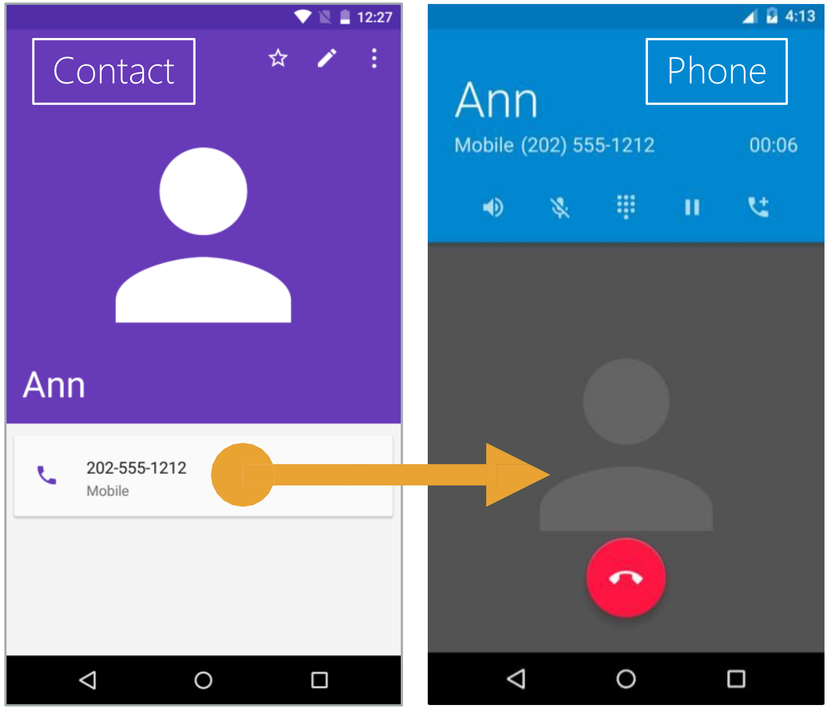
What is the back-stack?

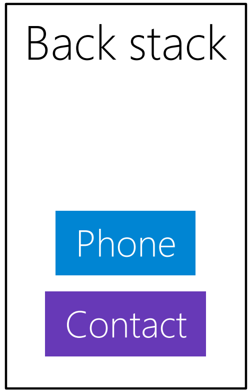
❖ The *back-stack* is a historical record of the user's live Activities



Back-stack scope

- The Activities in the back-stack may span multiple apps



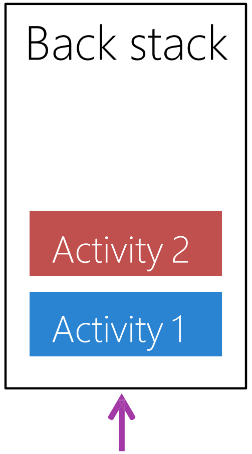


Stack contains Activities from Contacts and Phone

Back-stack push

- Android pushes Activities onto the back-stack automatically when you start them

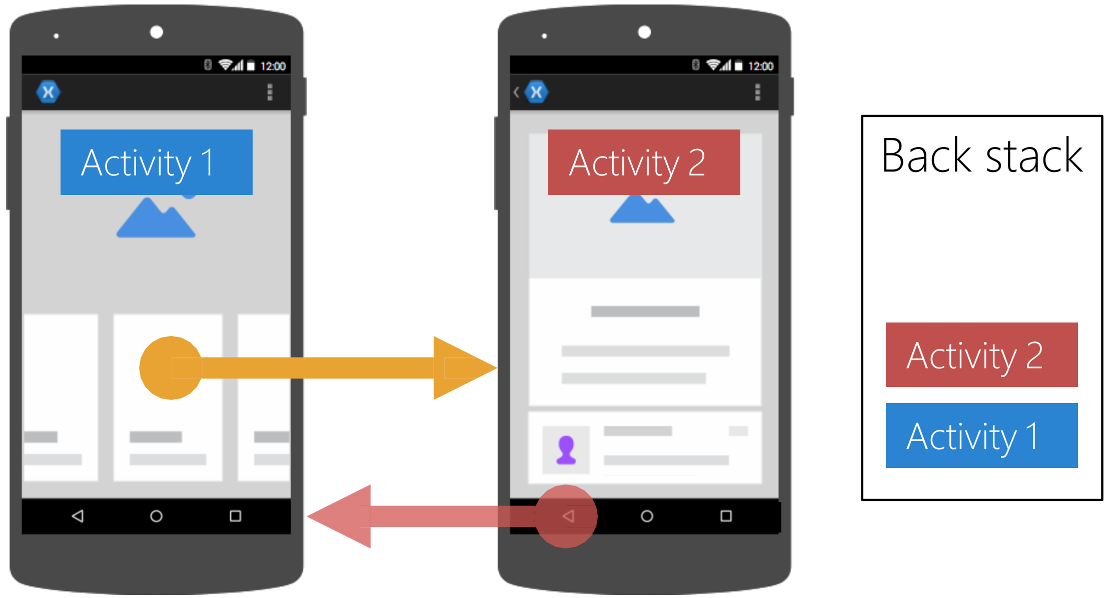
```
public class Activity1 : Activity
{ ...
  void OnClick(object sender, EventArgs e)
  {
    base.StartActivity(typeof(Activity2));
  }
}
```



Started Activities go on the stack

Automatic back-navigation

- The Back-button automatically pops the back-stack and returns the user to the previous Activity



Programmatic back-navigation

❖ Activity provides a **`Finish`** method that ends the current Activity and returns to the previous Activity on the back-stack

Ends the current Activity

```
public class Activity : ...
{ ...
  public virtual void Finish();
}
```

When to call Finish?

❖ An Activity can call **`Finish`** in cases when the behavior of the Back Button might be unclear to the user

E.g. add a "cancel" button to your UI so the user can be sure their changes will not be saved. ⟶

```
public class Activity2 : Activity
{ ...
  void OnCancelClick(object sender, EventArgs e)
  {
    base.Finish();
  }
}
```

Group Exercise

Programmatically end an Activity

Summary

1. Understand Stack Navigation
2. See the behavior of the Back-button
3. Programmatically finish an Activity

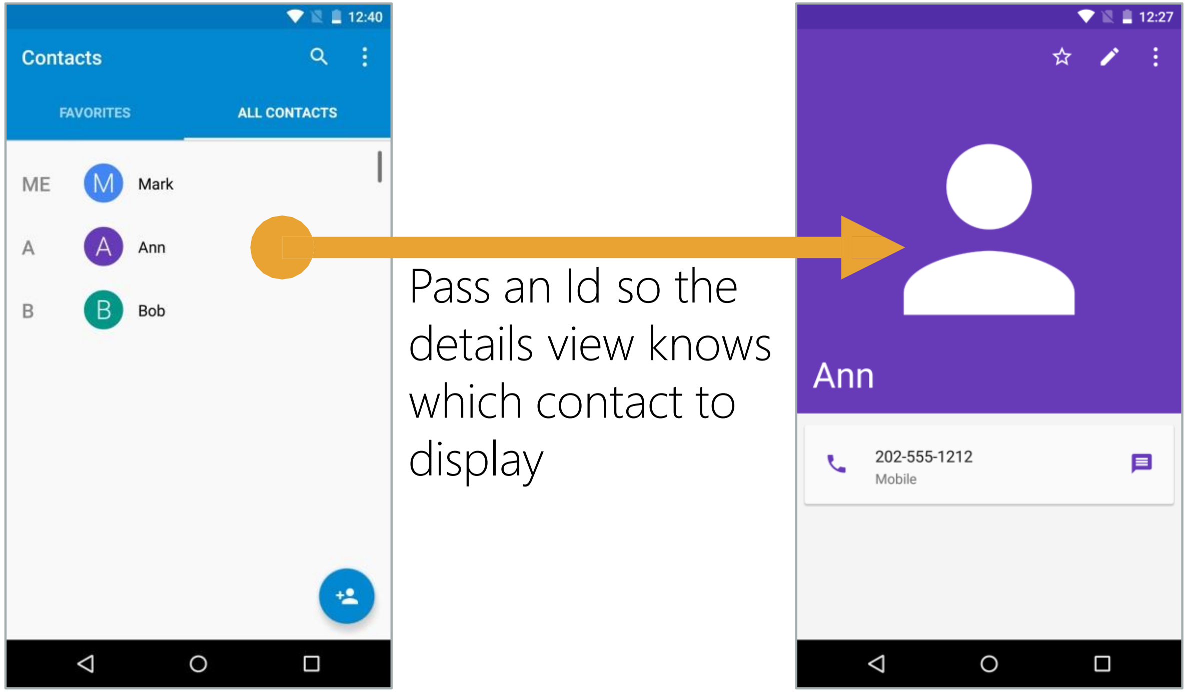
Pass arguments to an Activity

Tasks

1. Load a Bundle of arguments into an Intent
2. Retrieve the arguments in the target Activity

Motivation

❖ Activities typically need to pass data between them

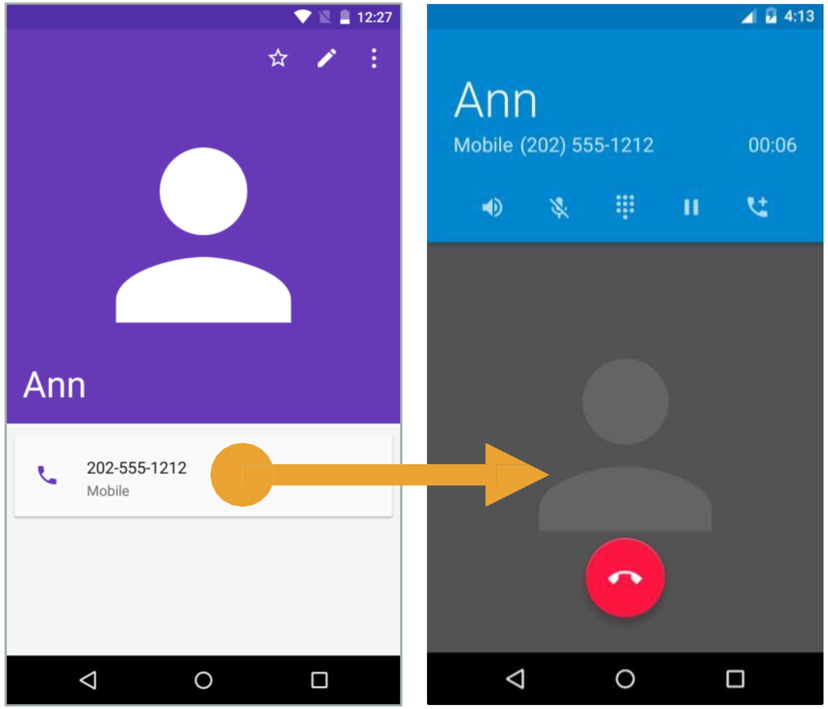


Pass an Id so the details view knows which contact to display

App process

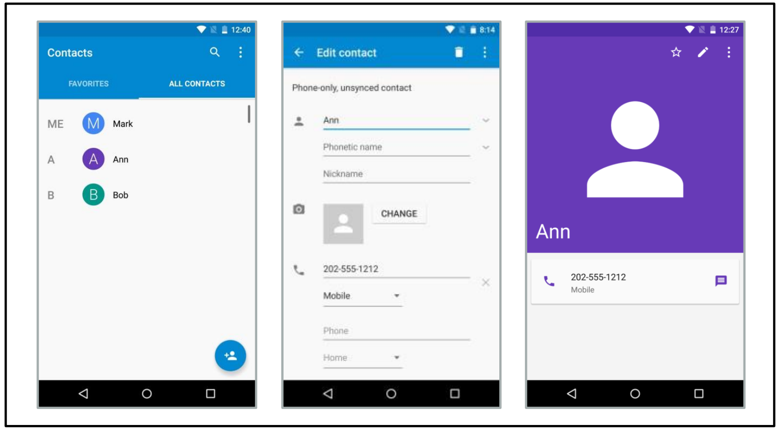
- Each app runs in its own process

The Contacts and Phone apps run in separate processes even when they work together

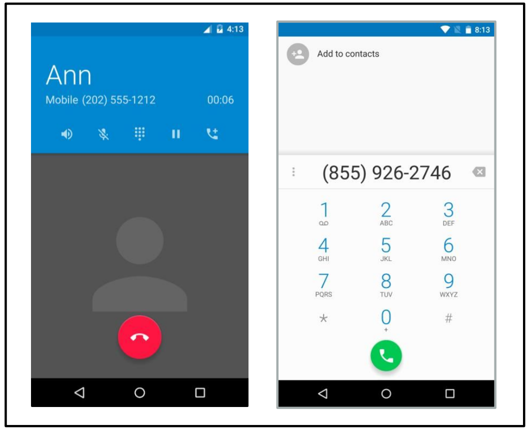


Activity process

- Each Activity runs in its app's process (i.e. the process associated with the app of which it is a part)



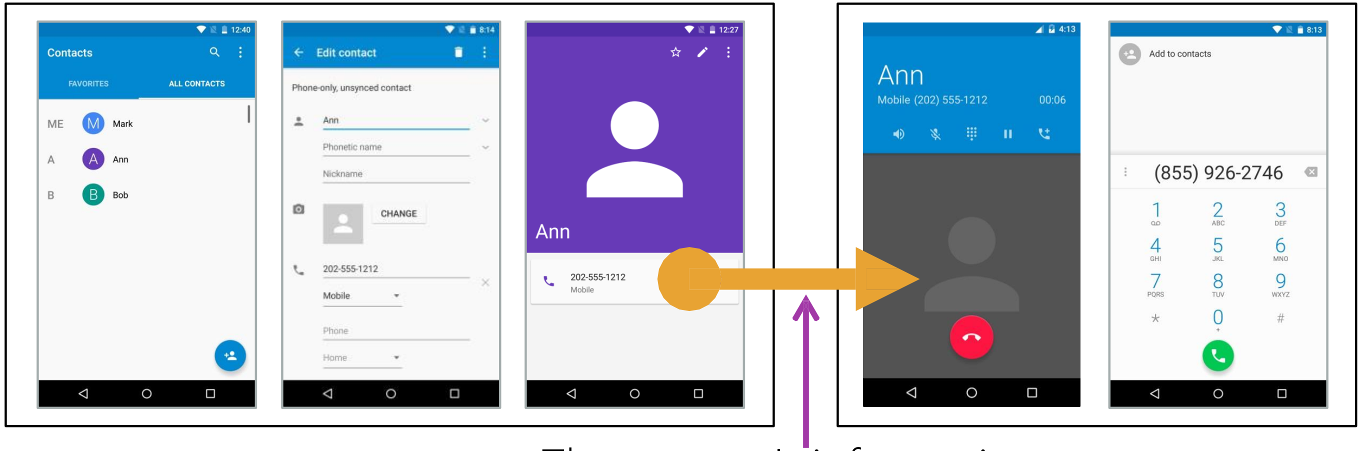
Contacts app process



Phone app process

Arguments and processes

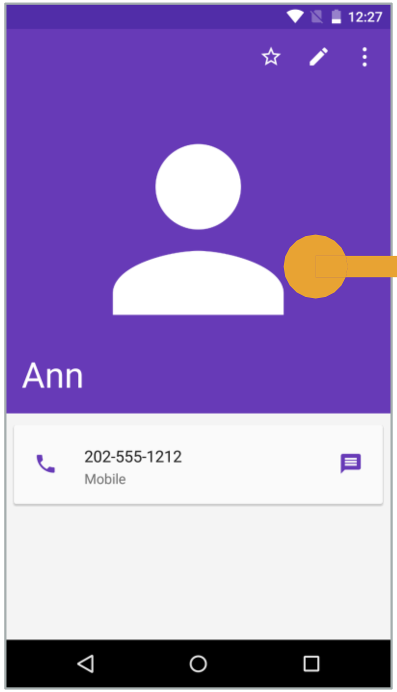
- Only simple types and serialized objects can move between Activities; object references cannot since they can't cross process boundaries



The contact's information moves between processes

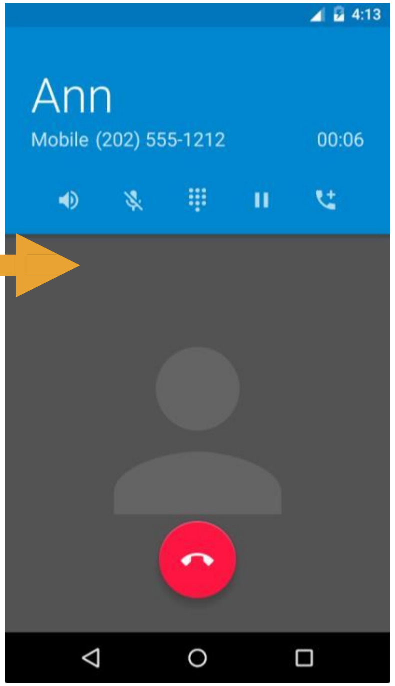
What is a Bundle?

❖ A *Bundle* is a collection of key ➡ value pairs passed between Activities



```
Name  →Ann
Phone→202-555-1212
Type  →Mobile
```

Keys are strings,
Values are limited
to a few types



Bundle and simple types

❖ Bundle has put/get methods for the simple types

Supports integer types, floating point types, Boolean, character, and string →

Also supports arrays and lists of the simple types (not shown) →

```
public sealed class Bundle : ...
{
  public void   PutInt   (string key, int    value);
  public int    GetInt   (string key, int    defaultValue);

  public void   PutDouble(string key, double value);
  public double GetDouble(string key, double defaultValue);

  public void   PutString(string key, string value);
  public string GetString(string key, string defaultValue);
  ...
}
```

Bundle and complex types

- Bundle supports two ways to serialize complex objects: **`Android.OS.IParcelable`** and **`Java.IO.ISerializable`**

Objects must be serialized to be stored in a Bundle →

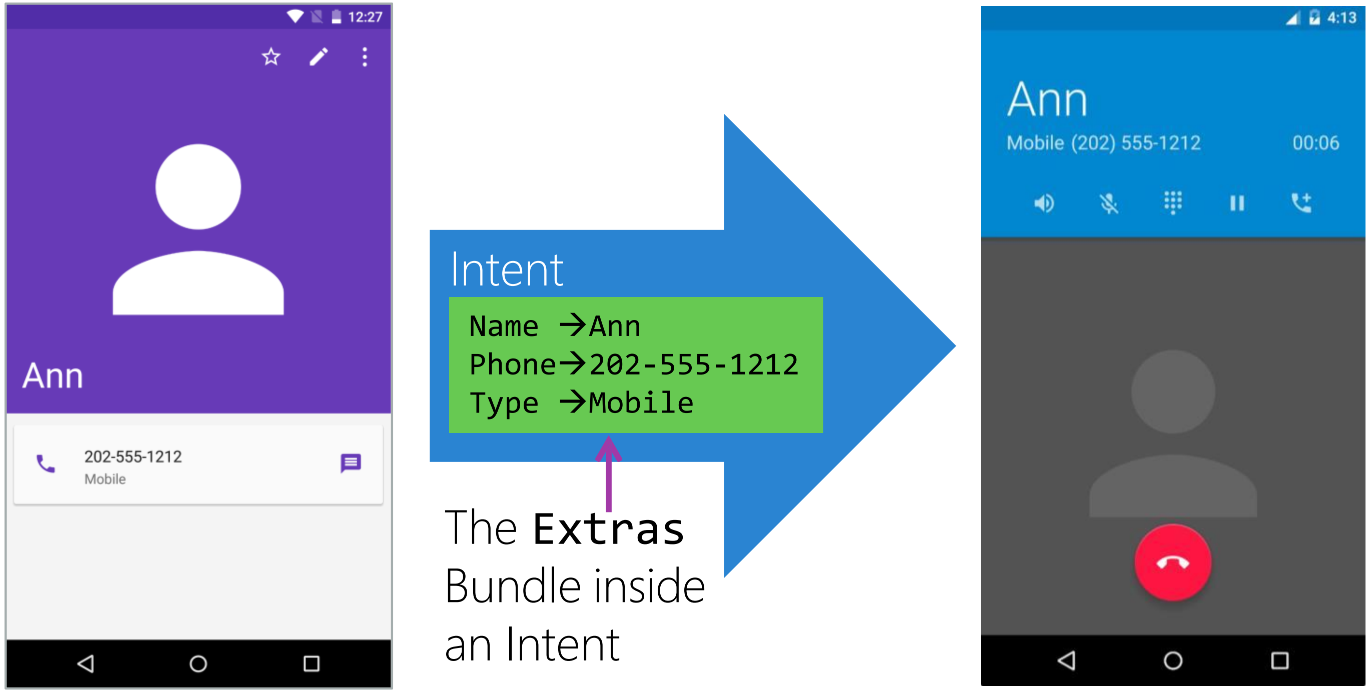
```
public sealed class Bundle : ...
{
  public void   PutParcelable(string key, IParcelable value);
  public Object GetParcelable(string key);

  public void          PutSerializable(string key, ISerializable value);
  public ISerializable GetSerializable(string key);
  ...
}
```

Xamarin has samples for how to implement both interfaces:
https://github.com/xamarin/monodroid-samples/blob/master/ExportAttribute/ExportAttributeTest/MainActivity.cs

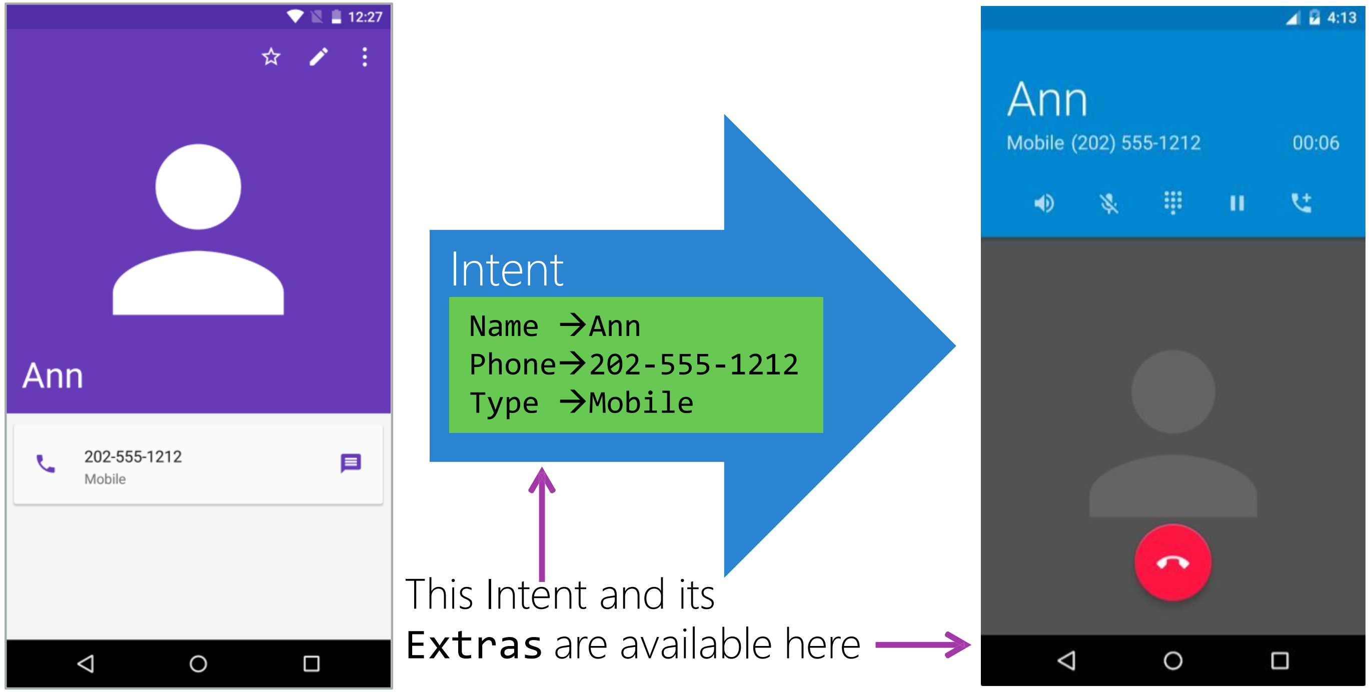
What are Intent Extras?

❖ `Extras` are a Bundle inside an Intent to be passed between Activities



Intent access in the Target

- The starting Intent is available in the Target's **`Intent`** property



This Intent and its **`Extras`** are available here

How to load Intent Extras

❖ There are two equivalent ways to load Intent Extras

Explicit creation →

```
var bundle = new Bundle();
bundle.PutInt("ContactId", 123456789);

var intent = new Intent();
intent.PutExtras(bundle);
```

Convenience methods →

```
var intent = new Intent();

intent.PutExtra("ContactId", 123456789);
```

How to retrieve Intent Extras

- There are two equivalent ways to retrieve Intent Extras in the Target

Explicit access →

```
int id = base.Intent.Extras.GetInt("ContactId", -1);
```

Convenience methods →

```
int id = base.Intent.GetIntExtra("ContactId", -1);
```

↑ Default value to be returned if key not found

Individual Exercise

Pass arguments to an Activity

Summary

1. Load a Bundle of arguments into an Intent
2. Retrieve the arguments in the target Activity

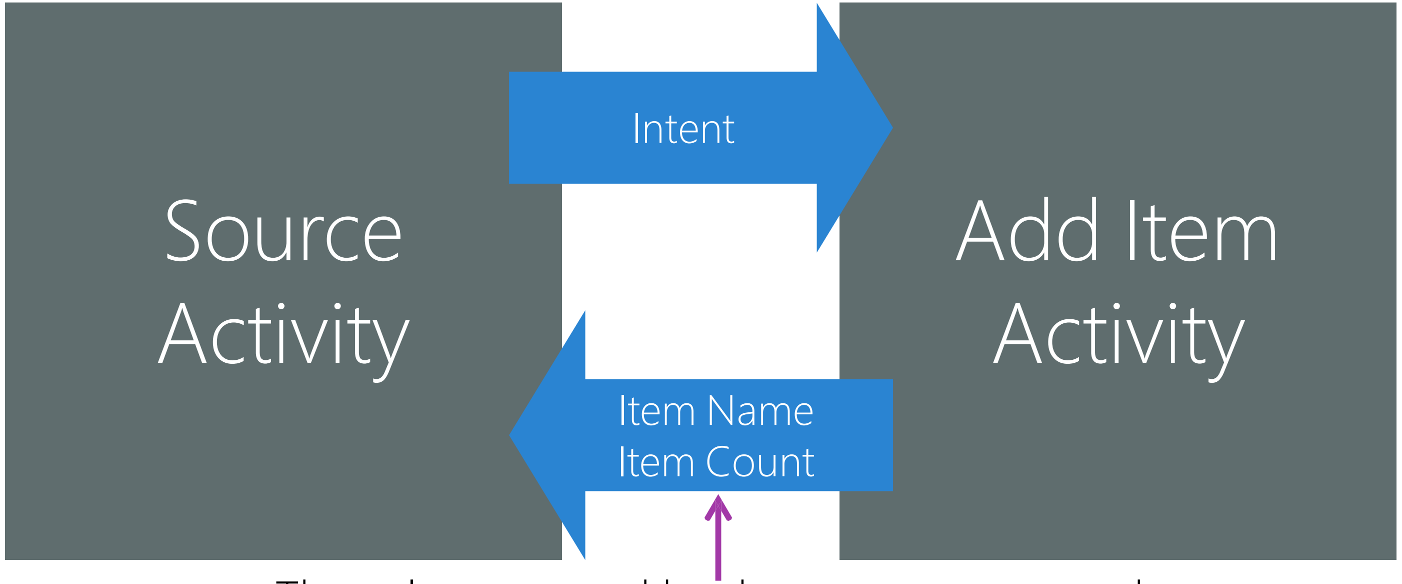
Get Activity results

Tasks

1. Pass a request code
2. Return a result code and Bundle
3. Retrieve results

Motivation

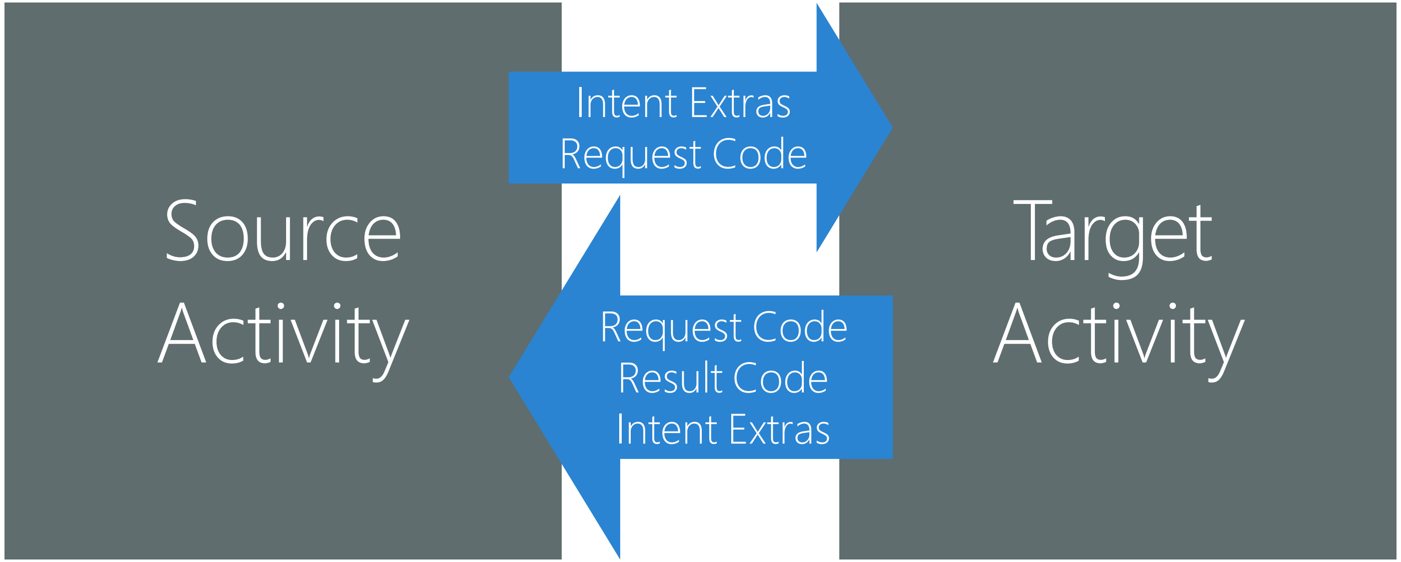
- An Activity often provides a service for another Activity and needs to report the results



The values entered by the user are returned

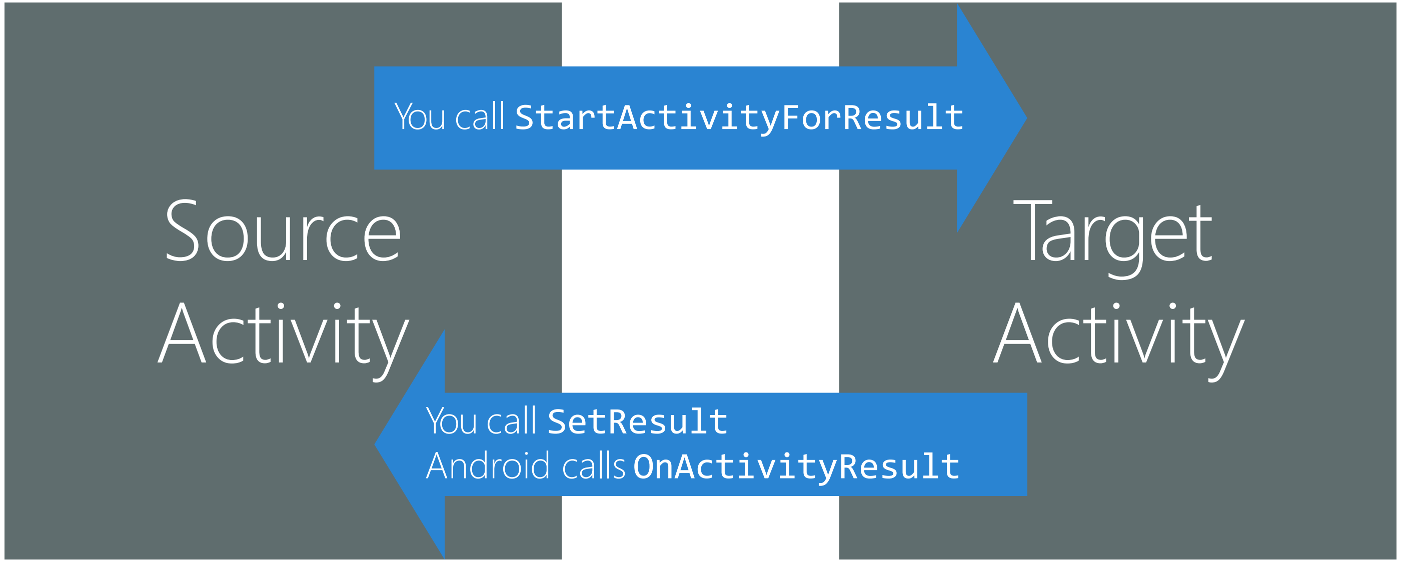
Data-flow overview

❖ Source and Target Activities pass several pieces of data between them



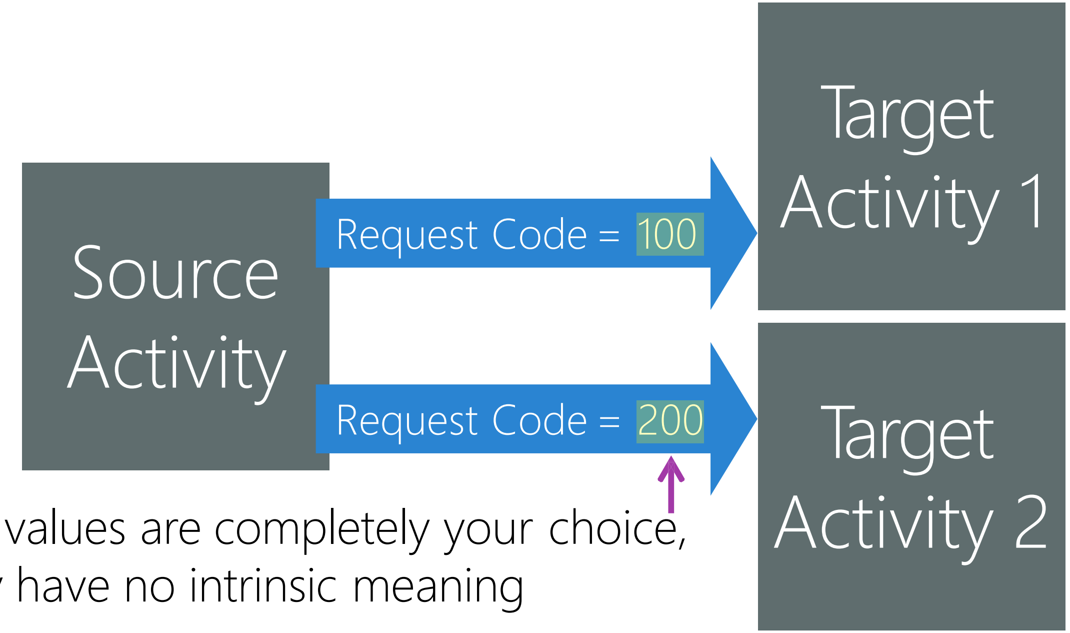
Method overview

❖ Source and target Activities use Activity methods to pass data


Source Activity
Target Activity
You call StartActivityForResult
You call SetResult
Android calls OnActivityResult

What is a request code?

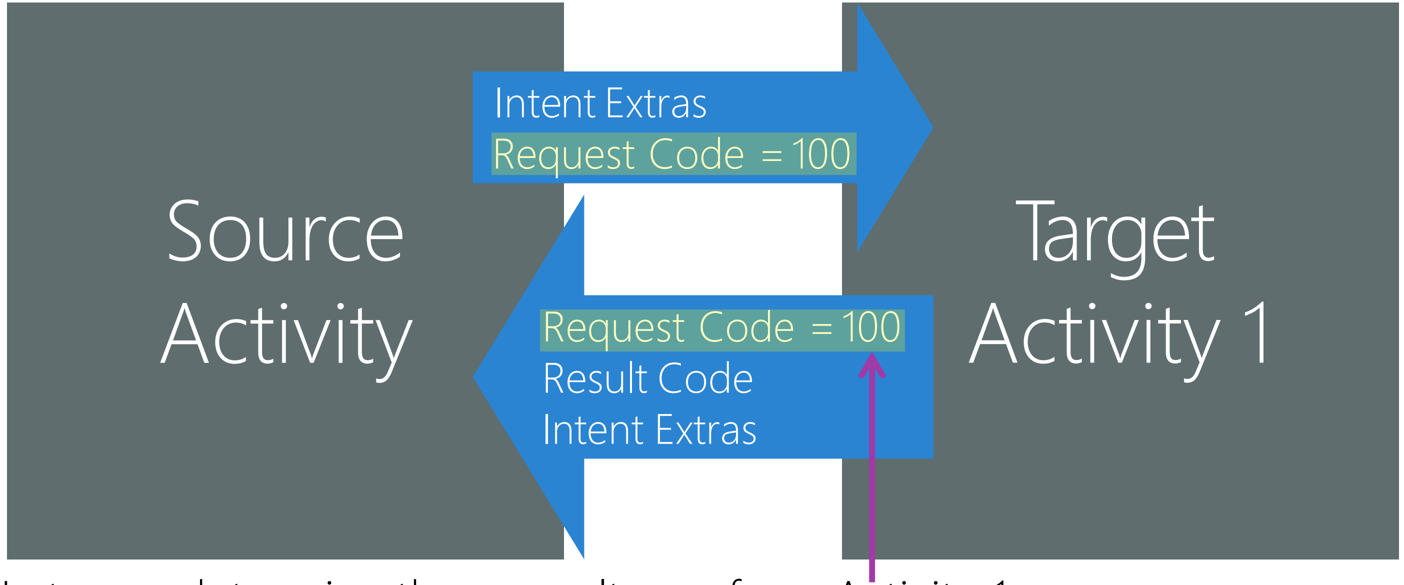
❖ A *request code* is an integer you pass to an Activity to help you identify it; you get that same value back when the Activity finishes



The values are completely your choice, they have no intrinsic meaning

Request code purpose

- All Activities report results via the same method in the Source; the request code is returned with the results to identify the Target



Lets you determine these results are from Activity 1

How to pass a request code

- Use **`StartActivityForResult`** to start an Activity and pass it a request code

```
public class Activity : ...
{
  public virtual void StartActivityForResult(Intent intent, int requestCode);
}
```

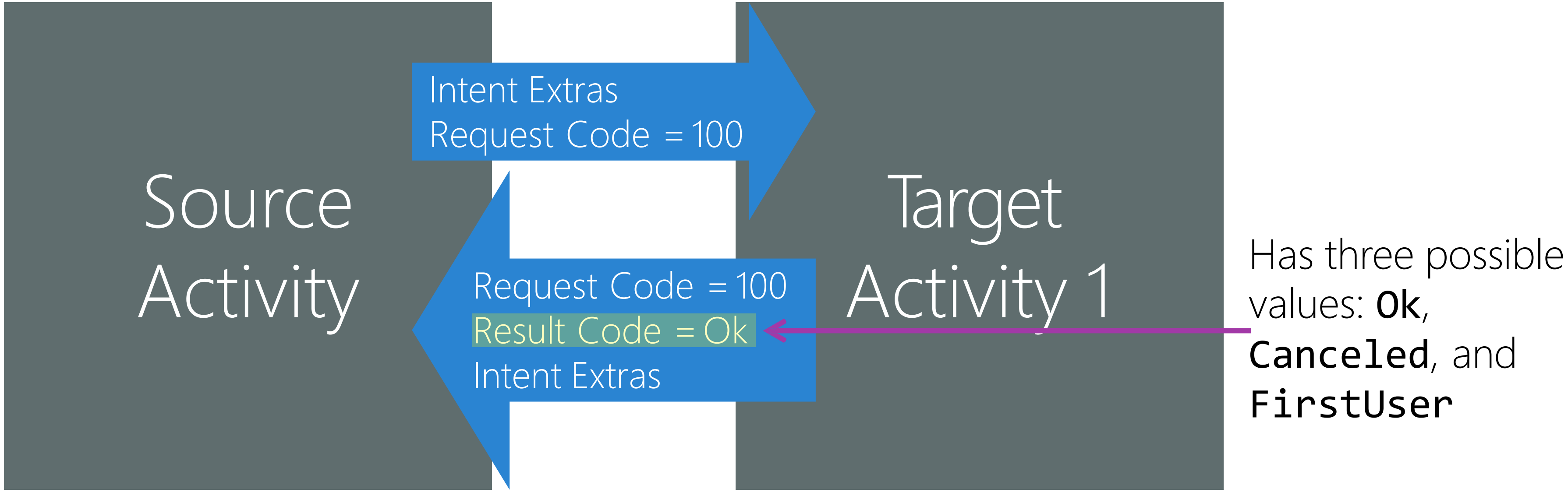
You call this in your Source Activity

Identifies the Target Activity to start and carries a Bundle of arguments if needed

Your choice of request code to let you track the Target

What is a Result code?

❖ A *result code* is an **`enum`** that an Activity uses to indicate success/failure

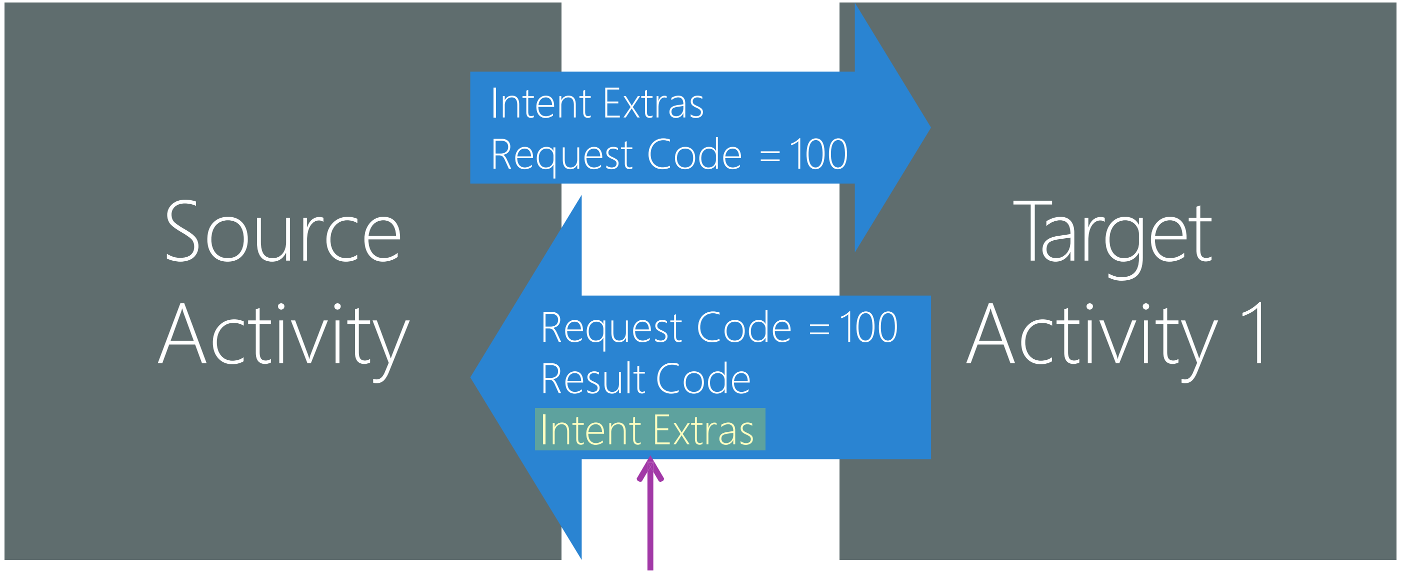


Has three possible values: **`Ok`**, **`Canceled`**, and **`FirstUser`**

`FirstUser` indicates the first integer value available for user-defined result codes (i.e. all predefined members have values less than **`FirstUser`**).

Result data

❖ An Activity can return a **`Bundle`** to the Activity that started it



You create an **`Intent`** and a **`Bundle`**, then load the Bundle with data

How to report results

- The Target Activity uses **`SetResult`** to specify what to return to the Source

```
public class Activity : ...
{ ...
  public void SetResult(Result resultCode);
  public void SetResult(Result resultCode, Intent data);
}
```

Target can report just a result code or a result code + data

How to retrieve results

❖ The Source Activity overrides **`OnActivityResult`** to receive results

```
public class SourceActivity : ...
{ ...
  protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
  {
    if (resultCode == Result.Ok && requestCode == 100)
    {
      string name  = data.GetStringExtra("ItemName");
      int    count = data.GetIntExtra   ("ItemCount", 0);
      ...
    }
  }
}
```

Data returned by the Target Activity

The Intent loaded by the Target Activity

Individual Exercise

Get Activity results

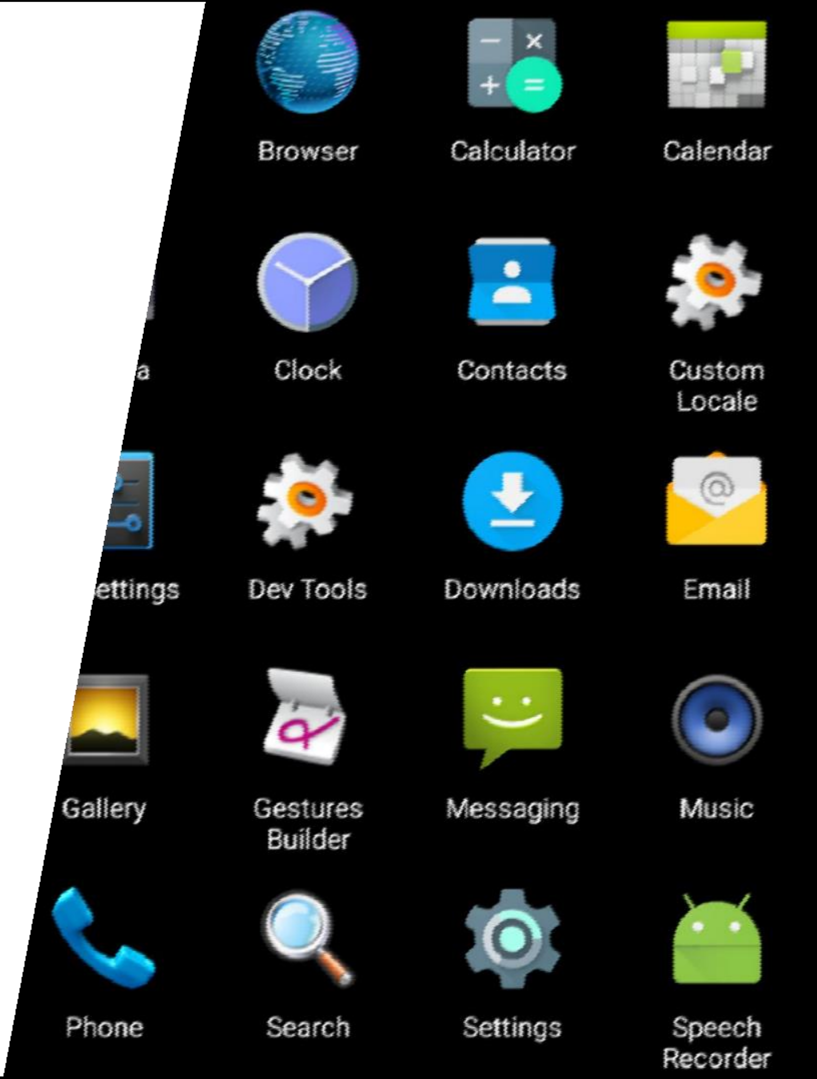
Summary

1. Pass a request code
2. Return a result code and Bundle
3. Retrieve results

Launch a system Activity

Tasks

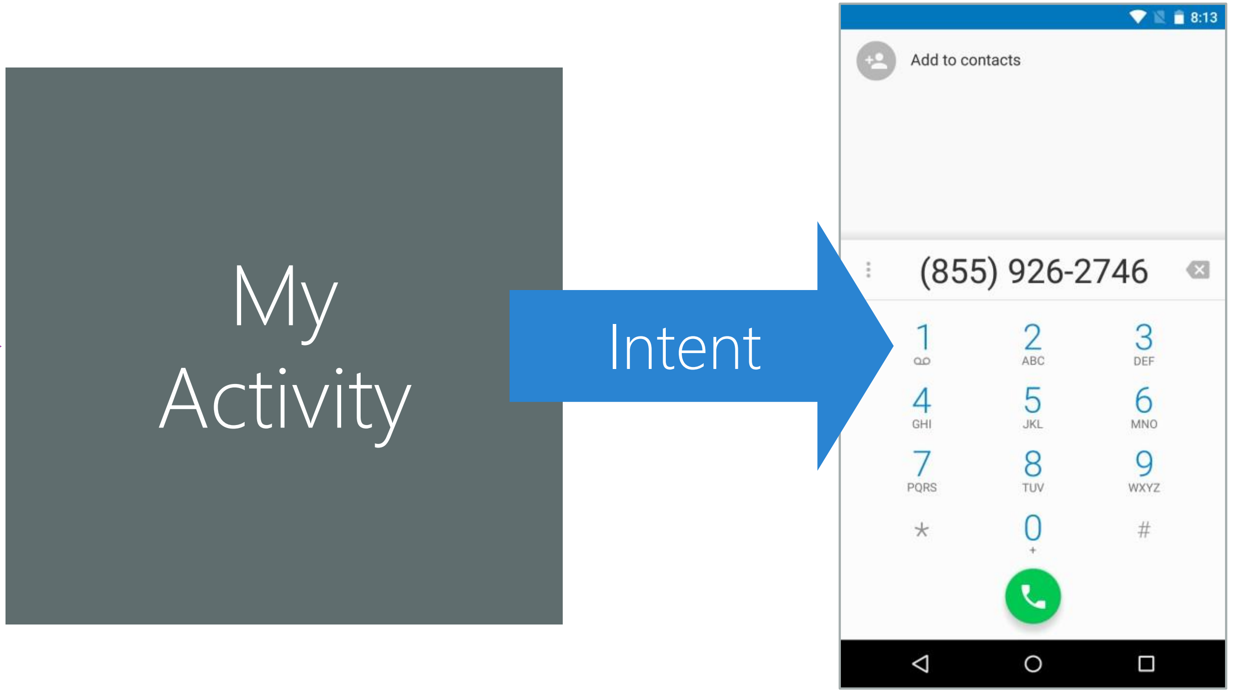
1. Create an implicit Intent
2. Load Intent Action, Data, and Extras
3. Verify that Android found an Activity that matches your implicit Intent



Motivation

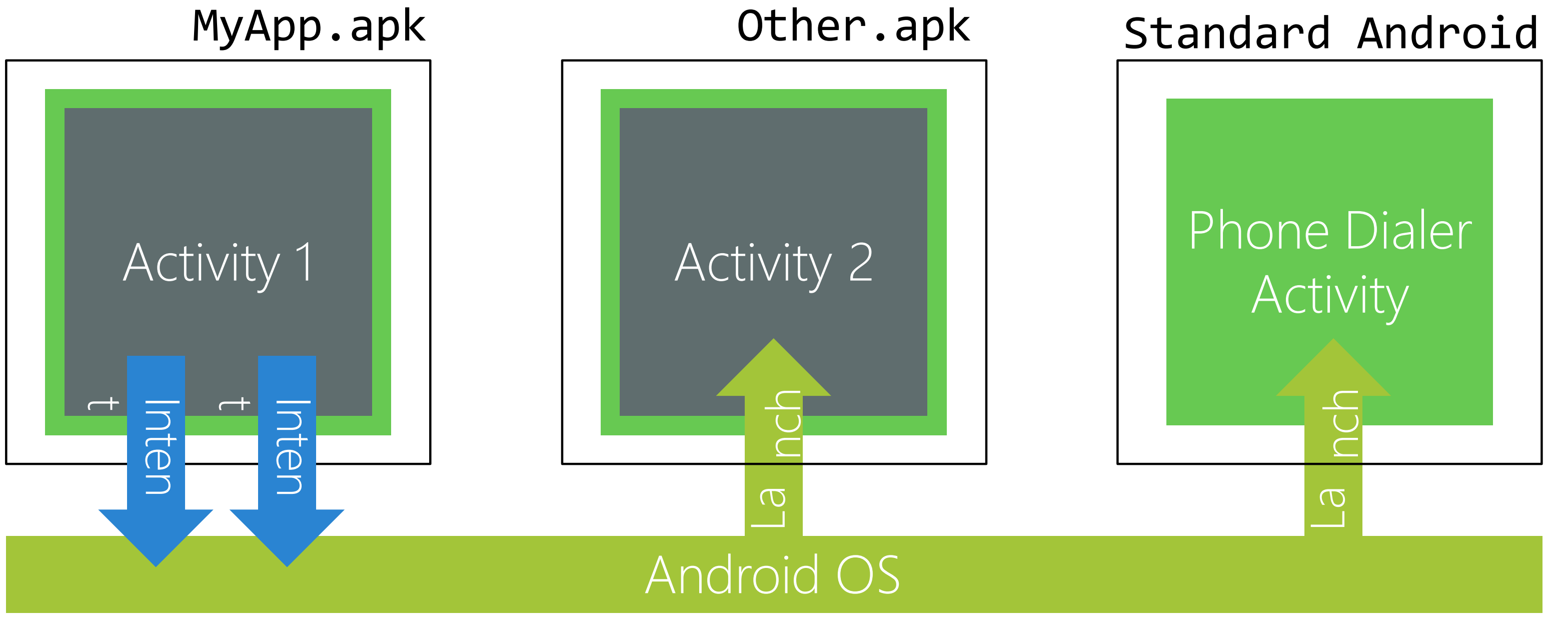
- You can utilize Android Activities like Contacts, Phone, Camera, etc.

E.g. your app could let the user call your sales team or help line →



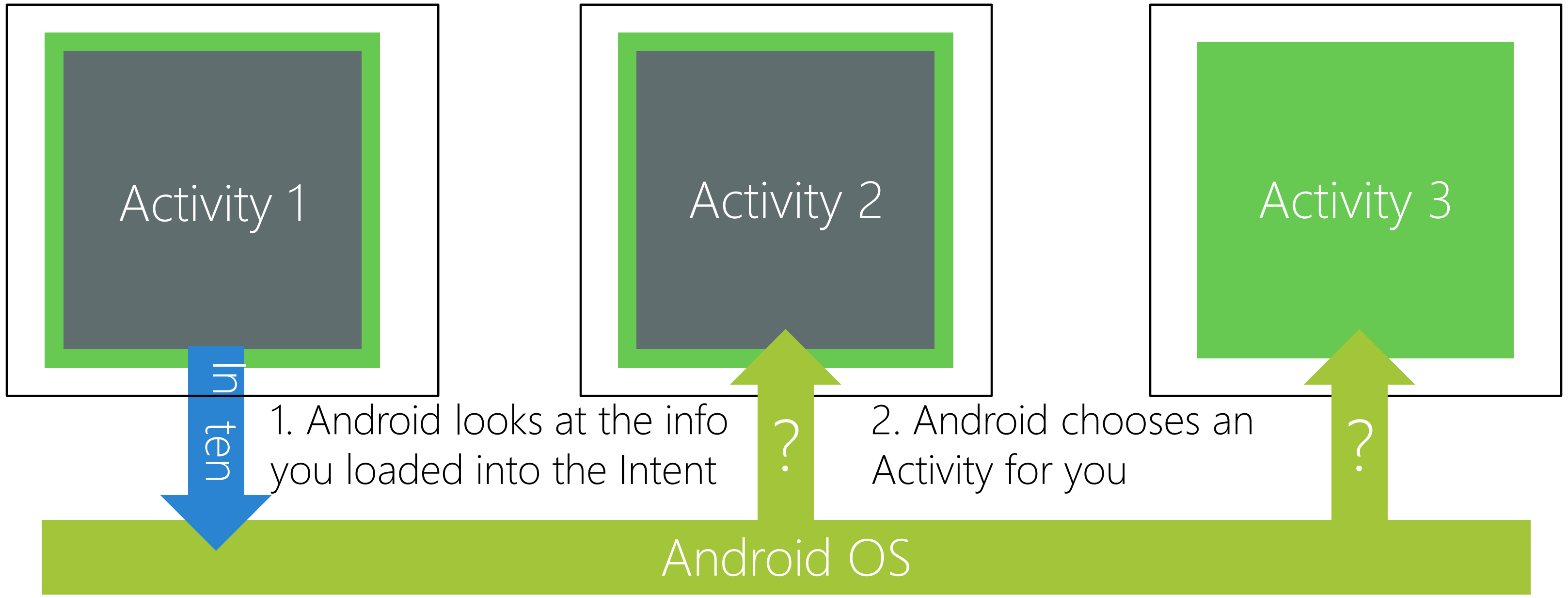
External collaboration

- You can start an Activity from a different .apk or one installed as part of a standard Android app



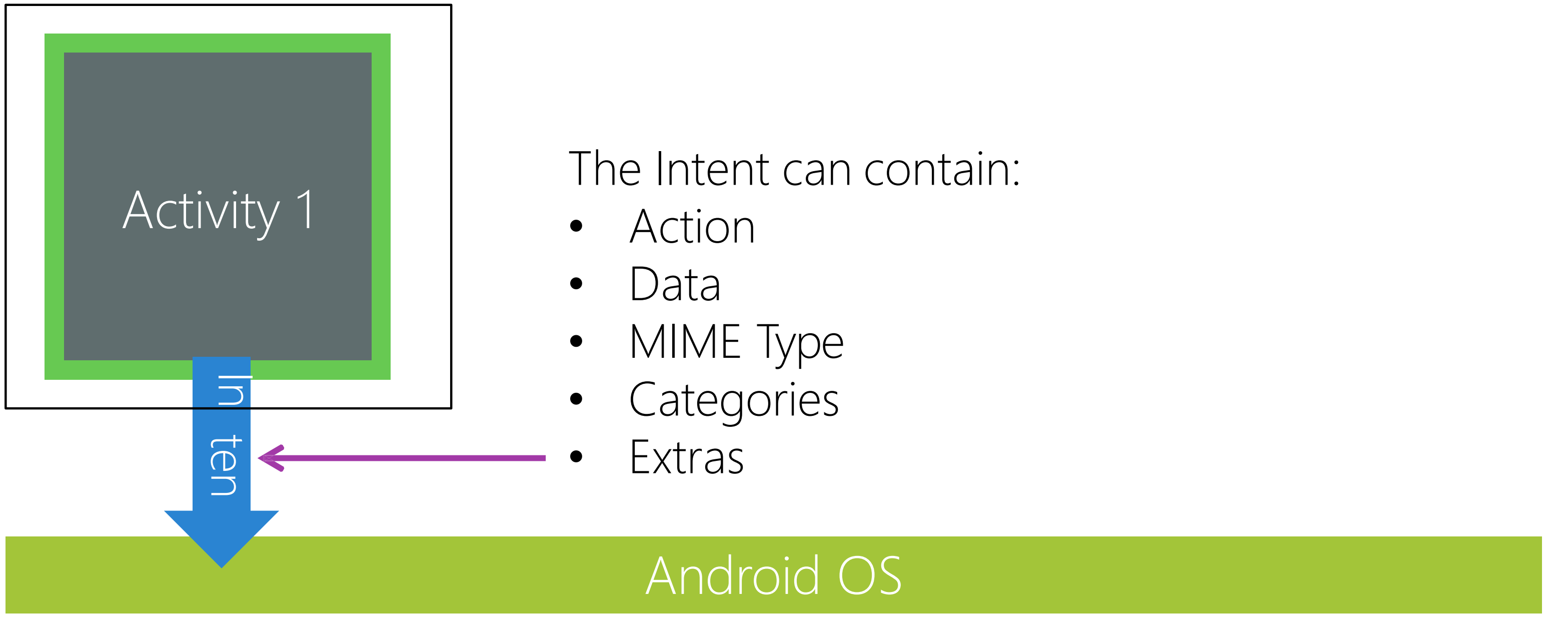
What is an implicit Intent?

❖ An *implicit Intent* describes what you want done without specifying which Activity should do it


Activity 1
Activity 2
Activity 3
Intent
1. Android looks at the info you loaded into the Intent
?
2. Android chooses an Activity for you
?
Android OS

Implicit Intent payload

❖ You load several pieces of information into an Implicit Intent that describe the operation you need performed



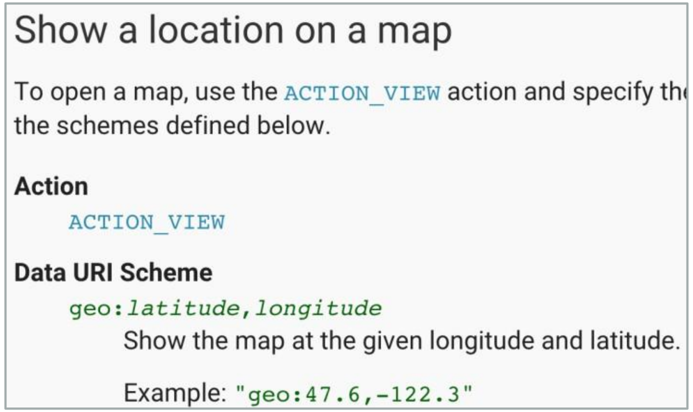
The Intent can contain:

- Action
- Data
- MIME Type
- Categories
- Extras

Xamarin University

How to know what to provide?

- The Android documentation tells you what to load into an Intent



Show a location on a map

To open a map, use the `ACTION_VIEW` action and specify th
the schemes defined below.

Action

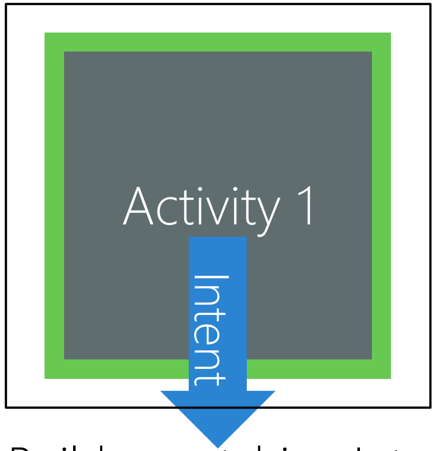
`ACTION_VIEW`

Data URI Scheme

`geo:`*`latitude,longitude`*
Show the map at the given longitude and latitude.

Example: `"geo:47.6,-122.3"`

1. Read the documentation



2. Build a matching Intent

How to create Intents for many common cases is described here:
https://developer.android.com/guide/components/intents-common.html

What is an Intent Action?

- An Intent *Action* specifies the type of work you need done

"Display some info"

"Dial the phone"

"Send a message"

Action specification

- Actions are specified using strings; the Intent class has a predefined string for many common Actions

Symbolic constant	Value	Meaning
`Intent.ActionView`	`android.intent.action.VIEW`	Show some info to the user
`Intent.ActionDial`	`android.intent.action.DIAL`	Dial the phone
`Intent.ActionEdit`	`android.intent.action.EDIT`	Let the user edit some data
`Intent.ActionSendto`	`android.intent.action.SENDTO`	Send a message
…	…	…

Some Action constants are packaged with the classes they are associated with. For example, you use `MediaStore.ActionImageCapture` to take a photo.

How to set the Action

- You can set an Intent's Action with either the constructor or the `SetAction` method

```
var intent = new Intent();

intent.SetAction(Intent.ActionView);
```

$\uparrow$

Action is a string, typical to use the predefined constants

What is Intent Data?

- Intent *Data* is a single piece of information for use by the Target Activity

Data for a map Activity →	`geo:37.797776,-122.401881?z=16`
Data for a phone dialer Activity →	`tel:(855) 926-2746`
Data for a browser Activity →	`http://www.xamarin.com`

The Android documentation will generally tell you what to use for the Data

How to set the Data

- Use the **SetData** method to load Data into an Intent

```
var intent = new Intent();
...
intent.SetData(Android.Net.Uri.Parse("http://www.xamarin.com"));
```

↑

Data is an Android URI

Xamarin University

What is Intent MIME Type?

❖ The MIME Type indicates the type of the Data you want the Intent to manipulate, it helps Android determine which Activity to launch

Insert a new contact →	`vnd.android.cursor.dir/contact`
Add a calendar event →	`vnd.android.cursor.dir/event`
Select an image →	`image/*`

The Android documentation will generally tell you what to use for the MIME Type

How to set the MIME Type

❖ Use the **`SetType`** method to set the MIME Type

```
var intent = new Intent();
...
intent.SetType("image/jpeg");
```

↑

Specify you want an Activity
that can work with jpeg images

Xamarin University

What is an Intent Category?

- A *Category* restricts the kind of Activity you would like to handle your Intent

Preference
(i.e. settings panel)

Tab
(i.e. intended to live inside a tab)

Openable
(i.e. picker)

You will not need to use Categories to launch most common Activities.

How to add a Category

❖ Use the **`AddCategory`** method to add one or more Categories

```
var intent = new Intent();
...
intent.AddCategory(Intent.CategoryPreference);
```

↑

The **`Intent`** class has constants for the standard Categories

Extras specification

- Extras are specified using strings; a few predefined strings are in the Intent class but most are packaged in the classes they work with

Symbolic constant	Value	Meaning
`Intent.ExtraEmail`	`android.intent.extra.EMAIL`	List of addresses for an email
`MediaStore.ExtraOutput`	`output`	Location for camera to save
`AlarmClock.ExtraRingtone`	`android.intent.extra.alarm.RINGTONE`	Tone to play for an alarm
`EventsColumns.Title`	`title`	Calendar event title
…	…	…

Xamarin University

Example: show a location on a map

- Use an implicit Intent with **`ActionView`** to show a map location

```
var intent = new Intent();

intent.SetAction(Intent.ActionView);

intent.SetData(Android.Net.Uri.Parse("geo:37.797776,-122.401881?z=16"));
```

↑ Latitude ↑ Longitude ↑ Zoom level

This requires a mapping app to run. Use an emulator with the Google APIs installed.

Example: send an email

❖ Use an implicit Intent with **`ActionSendto`** to send an email

```
var intent = new Intent();

intent.SetAction(Intent.ActionSendto);


// tell Android to use only email apps to service this request
intent.SetData(Android.Net.Uri.Parse("mailto:"));

intent.PutExtra(Intent.ExtraEmail, new string[] { "hello@xamarin.com" });
intent.PutExtra(Intent.ExtraSubject, "How are you?");
```

The Extras support all common fields like To, CC, Subject, etc.

Error checking

- To avoid a runtime exception, you should verify that your implicit Intent is valid before calling **`StartActivity`**

```
var intent = new Intent();
...
if (intent.ResolveActivity(PackageManager) != null)
{
  StartActivity(intent);
}
```

Test if Android found a matching Activity

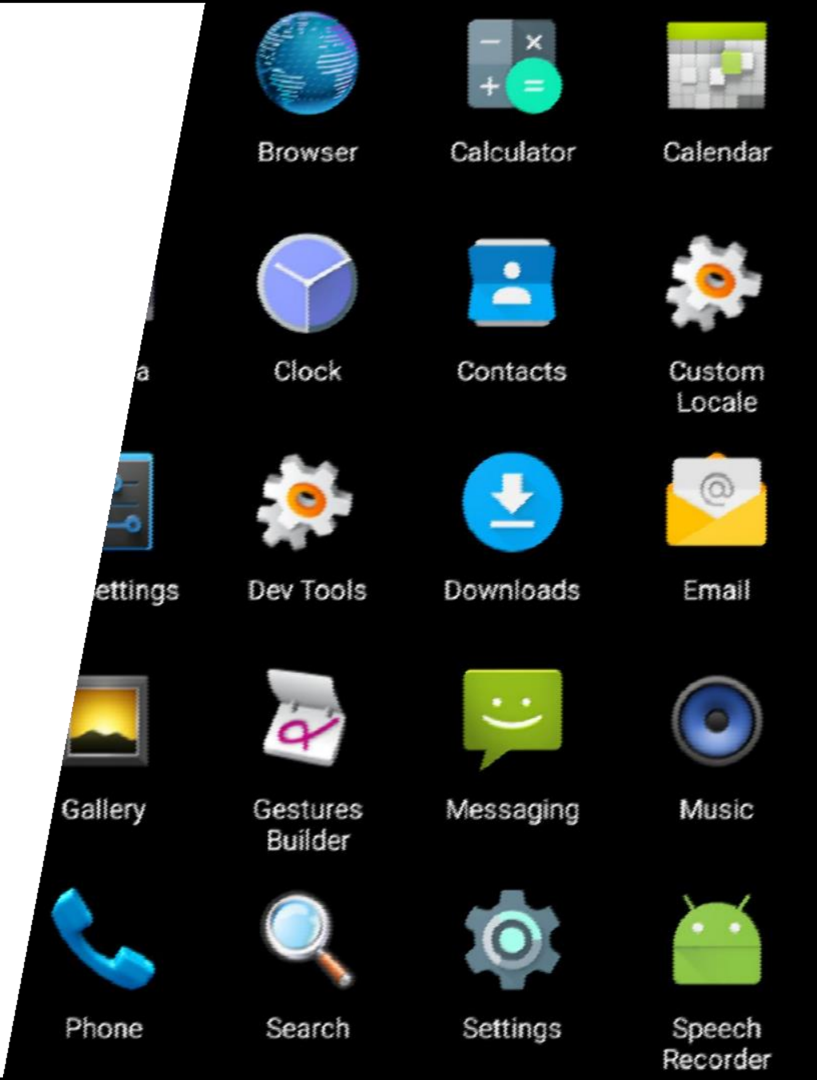
The Package Manager knows all Activities installed on the device. Your Activity inherited this property from **`Activity`**.

Group Exercise

Launch a system Activity

Summary

1. Create an implicit Intent
2. Load Intent Action, Data, and Extras
3. Verify that Android found an Activity that matches your implicit Intent



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile