

# Developing Android Applications with Kotlin: Getting Started

---

CREATING AND RUNNING A KOTLIN APP  
WITH ANDROID STUDIO



**Jim Wilson**

MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim   [blog.jwhh.com](http://blog.jwhh.com)



# What to Expect from This Course



- Create and run Kotlin app with Android Studio**
- Describe types with Kotlin**
- Activity code and layout interaction**
- Develop a multi-screen user experience**
- Options menus and action bar actions**
- Activity tasks and cross-app activity usage**
- Activity lifecycle**



# What to Expect from This Module



- Download and install Android Studio**
- Create Kotlin project in Android Studio**
- Modify activity layout with the designer**
- Add some code**
- Run app within an Android device emulator**



# Summary



## Android Studio

- Complete Android dev environment

## Layout designer

- Add views to the UX
- Set attributes of displayed views

## Accessing views from code

- Specify the value of the ID attribute
- Use variable with name matching the ID

## Android Emulator

- Run apps from your desktop
- a.k.a. Android Virtual Device (AVD)



# Resources and Adaptability

## **Resources externalize content from code**

- Maintained separately from source code
- Simplify adaptability

## **Android supports a rich device ecosystem**

- Apps need to adapt to device configuration differences



# Resources and Adaptability

## Hardware differences

- Screen size
- Screen density

## Dynamic configuration differences

- Portrait or landscape orientation
- Night mode on or off

## Setup and installation differences

- Locale
- Installed version of Android



# Resources and Adaptability

## **Dealing with differences can be challenging**

- Manual solutions are code intensive

## **Android resources handle differences**

- Can be associated with configurations
- Android handles selected appropriate resource for current configuration



# Drawable Resources

**Something that can be drawn to the screen**

- Images for menus, navigation drawer, status bar, notifications, etc.

**Vector**

- Scalable vector graphics
- Work well with simple graphics

**Raster**

- Non-scalable graphics
- \*.png preferred, can also use \*.jpg, \*.gif



# Drawable Resources

**Devices have variety of pixel densities**

- Presents challenge for raster graphics

**Raster graphics don't scale well**

- Stretch when scaled to higher densities



**Need different images for each density**



# Drawable Resources

## **Devices categorized by screen density**

- Separate drawable folder for each
- Create graphic files sized appropriately for each screen density



# Drawable Resources

Category	Pixels Per Inch	Drawable Folder
-		-



## Summary



**Resources externalize content from code**

- Simplify adaptability

**Android supports rich device ecosystem**

- Broad set of device characteristics

**Can associate resources with characteristics**

- Android handles details of selecting appropriate resource for device



# Summary



## **Some characteristics can change at runtime**

- Android destroys activity
- Recreates with appropriate resources

## **Characteristic specific layouts**

- Device orientation, screen size, etc.
- Be sure to keep view IDs consistent



# Summary



## Use string resources for UI text

- Simplifies multi-language support
- Devices have a language specific locale
- Can associate string resources with specific languages



# Summary



## **Include density specific drawables**

- Raster graphics don't scale well

## **Characteristic specific resource folders**

- Visible in “Project” view of Android Studio project window



# Drawable Resource

**Something that can be drawn to the screen**

- Often used with ImageView, ImageButton
- Icons for menus, navigation drawer

**Project location**

- res/drawable

**A variety of drawable types supported**



# Drawable Resource

## Raster

- Non-scalable graphics file
- \*.png preferred, can also use \*.jpg, \*.gif

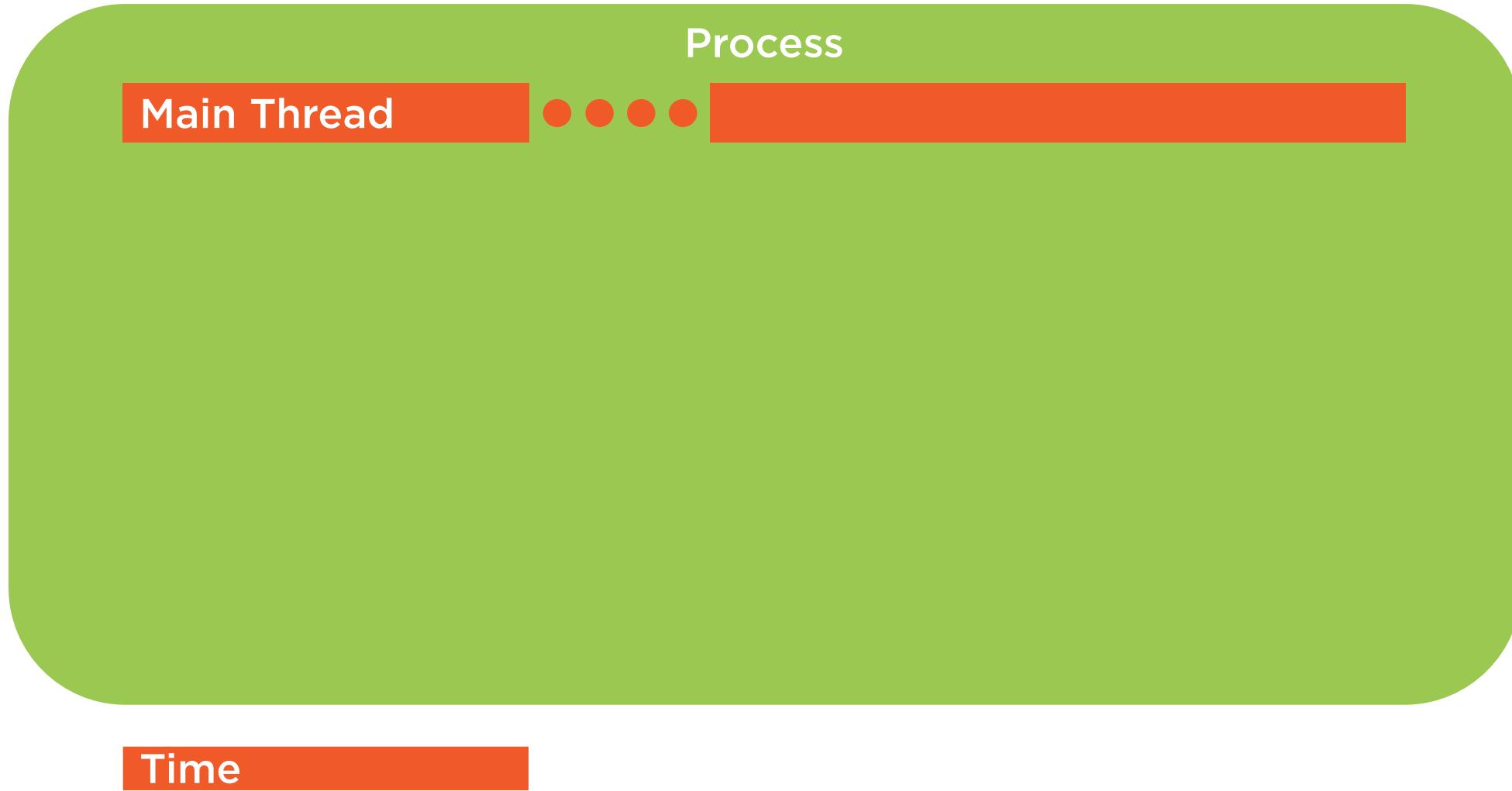
## Vector

- Scalable vector graphics file
- Work well for simple graphics
- Import with Vector Asset Studio





# Protecting the Main Thread



# Protecting the Main Thread

## Common operations to avoid

- Reading from “disk” storage
- Writing to “disk” storage
- Interacting with network

## Being certain can be challenging

- Programs are often complex
- Operations may unexpectedly occur



# Protecting the Main Thread

## **StrictMode class**

- Can detect undesirable operations
- Enforces a penalty when detected

## **Use during debugging/testing**

- Build desired thread policy
- Set policy at app/activity start



# Protecting the Main Thread

## Setting thread policy

- Use StrictMode.setThreadPolicy
- Accepts StrictMode.ThreadPolicy

## Creating ThreadPolicy

- Uses builder pattern
- StrictMode.ThreadPolicy.Builder



# Protecting the Main Thread

## **Set what you want to detect**

- detectDiskReads
- detectDiskWrites
- detectNetwork
- Commonly use detectAll

## **Decide what you want the penalty to be**

- penaltyLog
- penaltyException
- penaltyDialog
- penaltyDeath



# Understanding the Role of the Main Thread

## The main thread and app UI

- Most UI work performed on main thread
- Many programmatic UI operations only allowed to occur on main thread



# Doing Work on a Background Thread

## **Perform the long-running work**

- Runs on background thread

## **Present work results**

- May require interacting with UI
- Runs on main thread



# Doing Work with AsyncTask

## AsyncTask class

- Manages threading details
- Methods for each phase of work
- Methods run on appropriate thread

## Performing work with AsyncTask

- Define new class that extends AsyncTask
- Override appropriate methods



# AsyncTask Methods

## **doInBackground method**

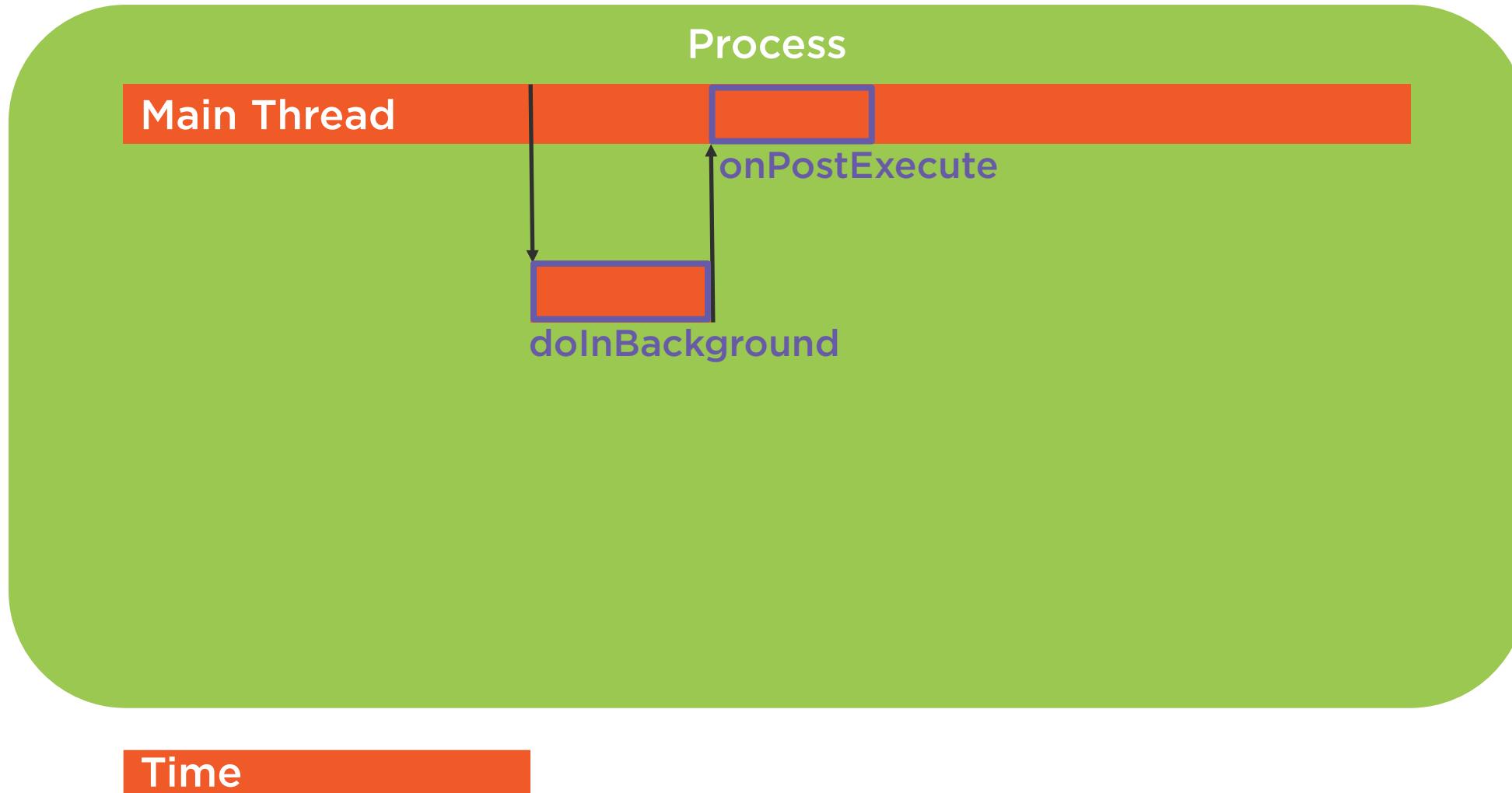
- Runs on background thread
- Add code to do the work

## **onPostExecute method**

- Runs on main thread
- Add code to present results



# AsyncTask Methods



# Passing Data Between AsyncTask Methods

## Initiating AsyncTask processing

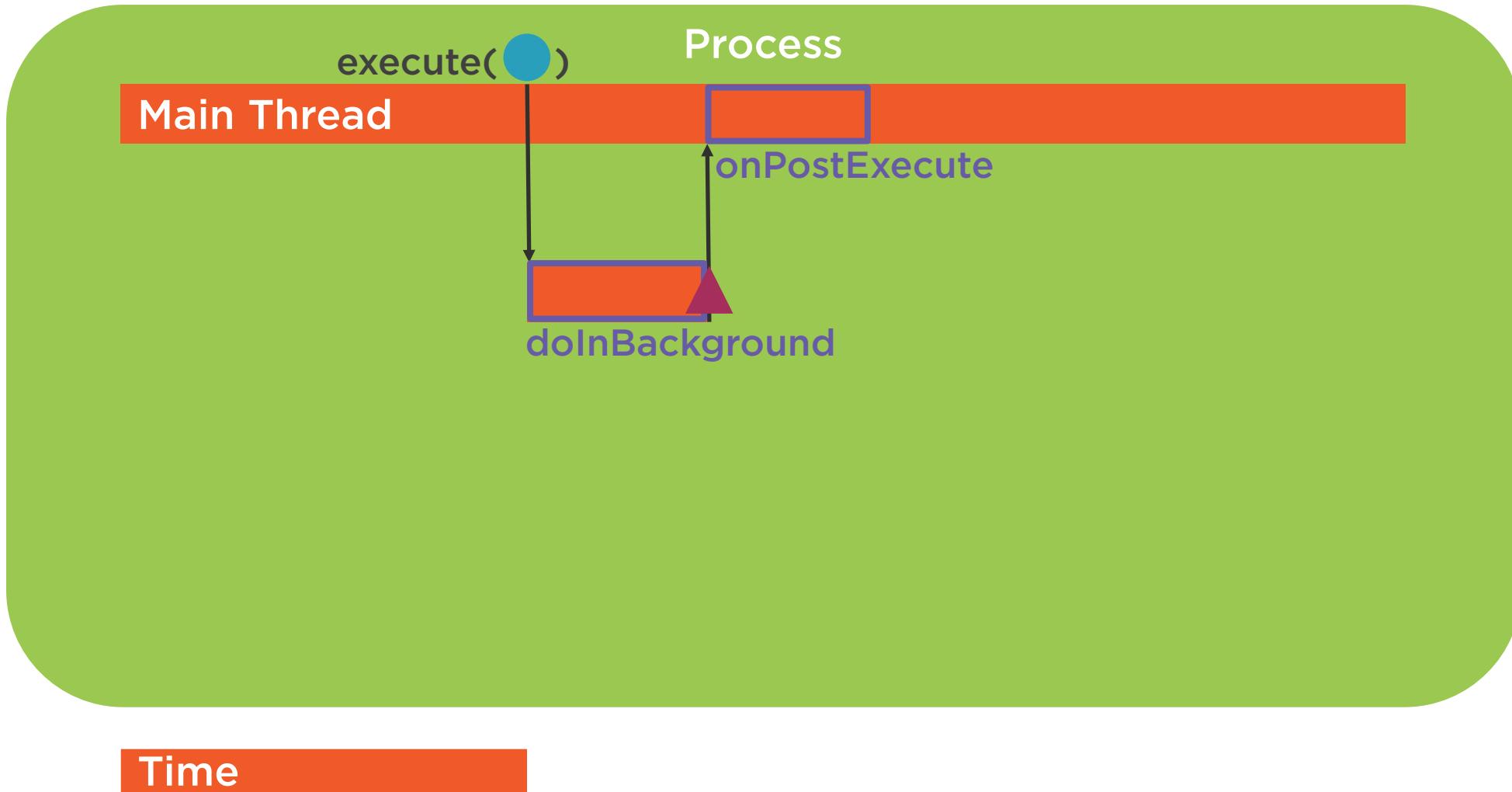
- Call execute method
- Accepts a variable length parameter list
- Parameters passed to doInBackground

## Providing background work results

- Return from doInBackground method
- Value passed to onPostExecute



# Passing Data Between AsyncTask Methods



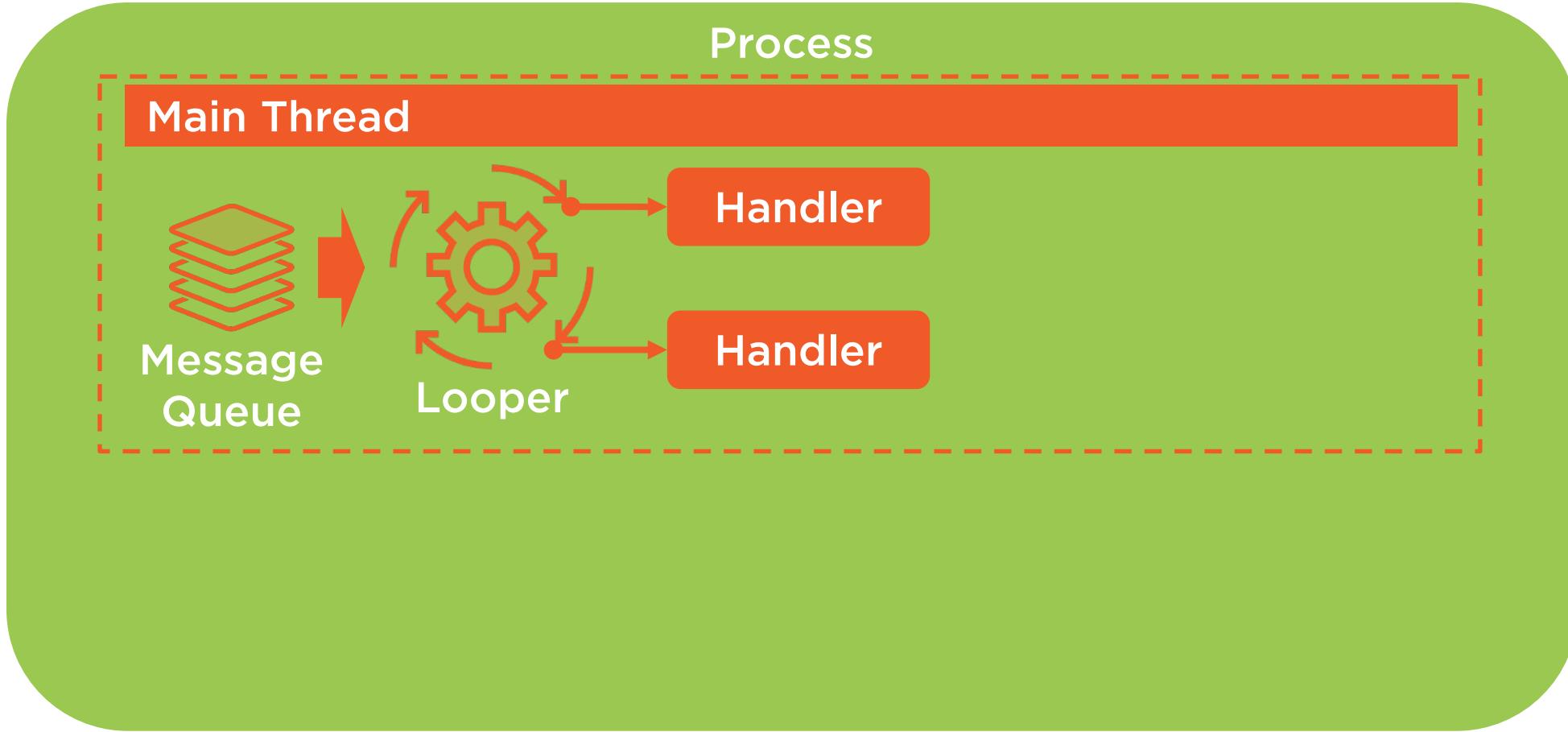
# Passing Data Between AsyncTask Methods

```
AsyncTask task = new AsyncTask< , , >() {  
    doInBackground(Type1... Params) {  
        Type3 result = // result of work  
        return result;  
    }  
    void onPostExecute(Type3 t3) { }  
};  
task.execute(/* Type1 values */);
```

The diagram illustrates the data flow between the `doInBackground` and `onPostExecute` methods of an `AsyncTask`. A red arrow points from the `result` variable in the `doInBackground` method to the `t3` parameter in the `onPostExecute` method. A blue arrow points from the `/* Type1 values */` placeholder in the `execute` call up to the `Params` type in the `AsyncTask` constructor.



# Working with Handlers



# Working with Handlers

## LooperThread

- Has a Looper and MessageQueue
- Work dispatched to Handler instances
- Main thread is a LooperThread
- Can create others if needed

## Handlers are our main point of interaction

- Handler instance enqueues work
- Handler instance receives the work when dispatched by Looper



# Working with Handlers

## Constructing a Handler

- Must be associated with a Looper
- By default uses current thread's Looper
- Can associate with main thread by constructing with Looper.getMainLooper



# Working with Handlers

## **Handler is bound to Looper**

- Can enqueue work from any thread
- Work always performed on thread of associated Looper

## **Primary uses of Handlers**

- Sending work to one thread from another
- Scheduling work for future execution



# Protecting the Main Thread

## Setting thread policy

- Use StrictMode.setThreadPolicy
- Accepts StrictMode.ThreadPolicy

## Creating ThreadPolicy

- Uses builder pattern
- StrictMode.ThreadPolicy.Builder



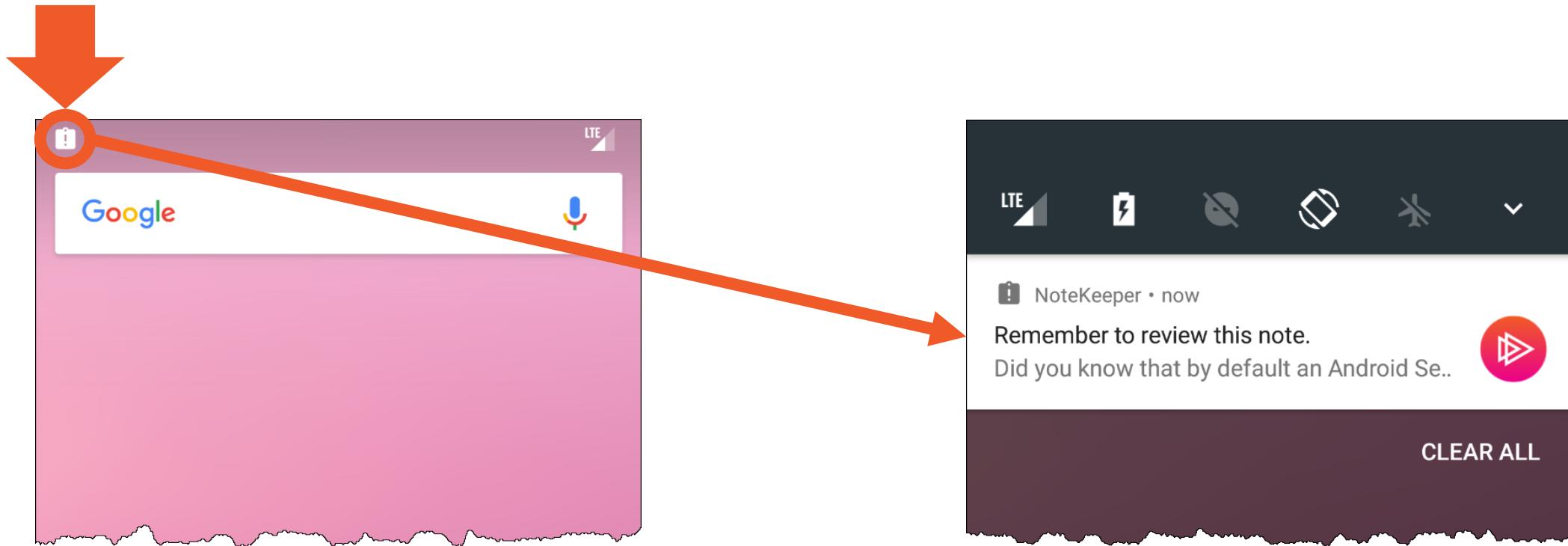
# Notifications

## **UI experience displayed outside your app**

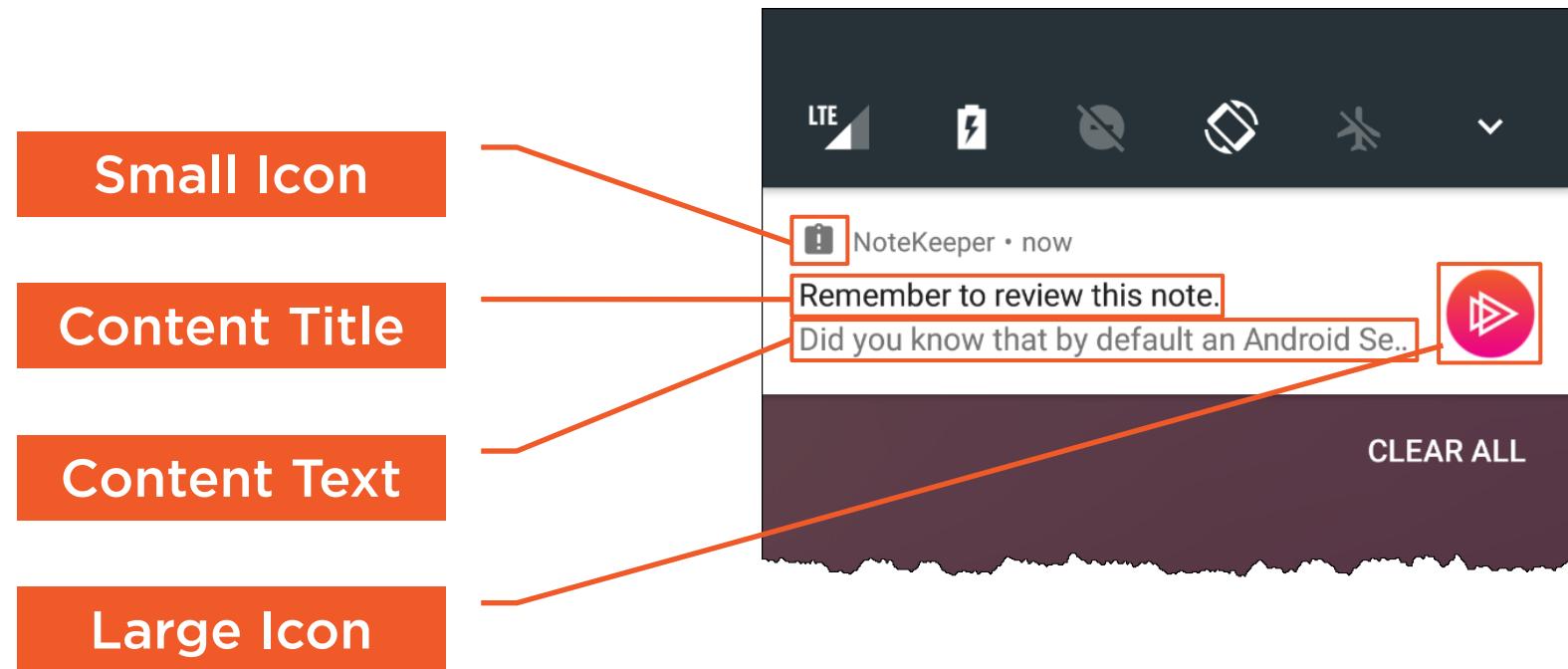
- Notification displayed by the system
- Initially shows as icon in notification area
- User opens notification drawer to view full notification



# Notifications



# Notifications



# Notifications

## Specific features vary by Android version

- Capabilities have evolved over time
- Manually managing version differences would be cumbersome

## We almost always use compatibility class

- NotificationCompat
- Methods available for all features
- Adjusts actual behavior to the Android version that the app is running on



# Creating and Displaying Notifications

## Notifications use builder pattern

- `NotificationCompat.Builder`
- Exposes “set” methods for each feature
- Once features set, call `build` method to create `NotificationCompat` instance



# Creating and Displaying Notifications

## **NotificationManager**

- Handles display management
- Access with Context.getSystemService

## **Android Studio generated helper class**

- Centralizes notification creation code
- Handles NotificationManager interaction
- Exposes static notify & cancel methods



# Creating and Displaying Notifications

## Displaying a notification

- Use notify method
- Modify to customize notification
- Subsequent notify calls replace existing notifications displayed by same class

## Removing a notification

- Explicitly remove with cancel method
- Normally set to automatically be removed when tapped



# Notification Styles

## **Allows specialization of notification**

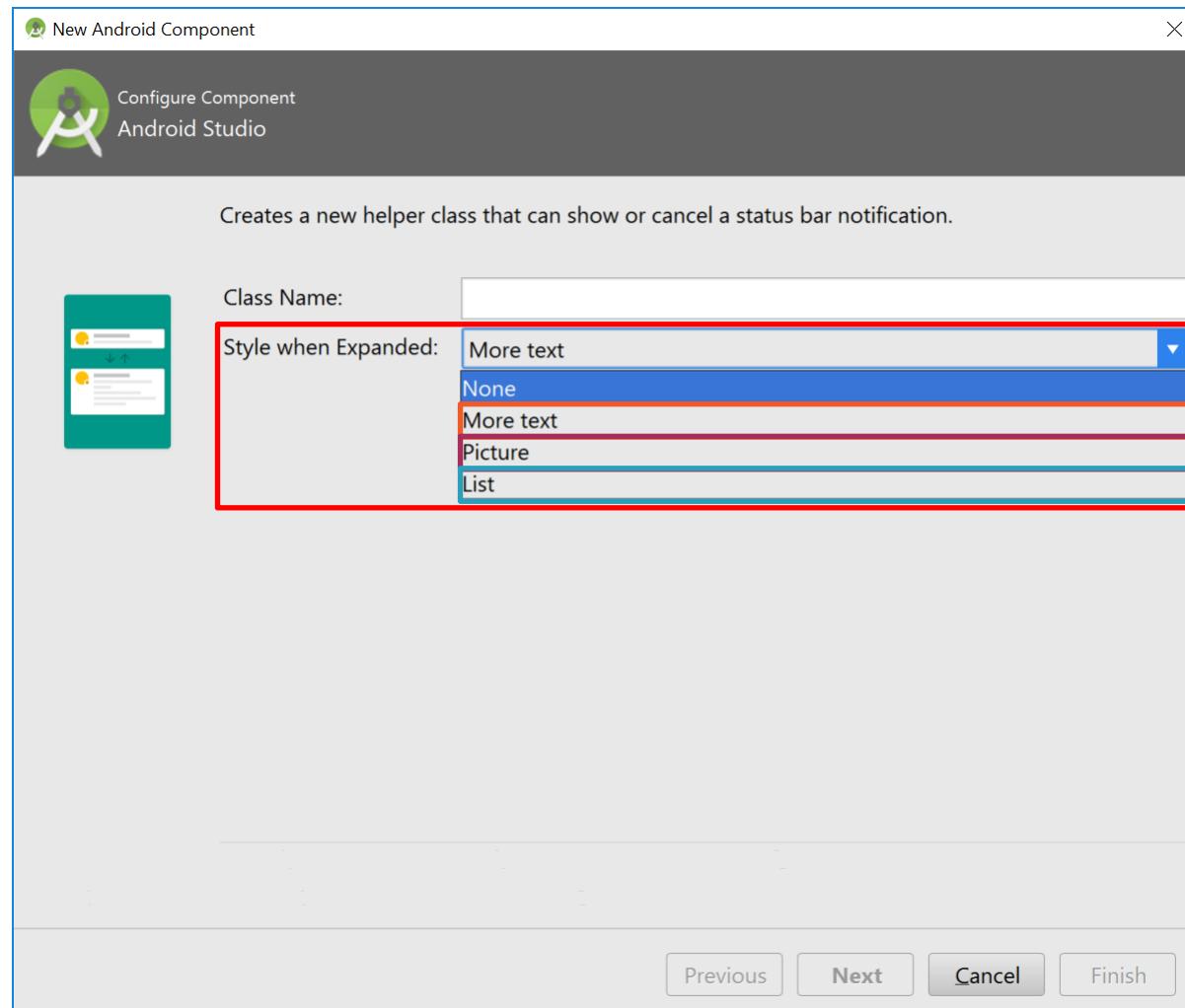
- Creates an expandable notification
- Each style has a different expanded appearance

## **Applying a style**

- Create instance of style type
- Set style features
- Pass instance to builder setStyle method



# Notification Styles



**BigTextStyle**

**BigPictureStyle**

**InboxStyle**



# Notification Styles

## **Some expanded content same**

- Can add text to summary line
- Can provide different title from regular content title



# Notification Styles

## **BigTextStyle**

- Provides ability to display a relatively large amount of text

## **BigPictureStyle**

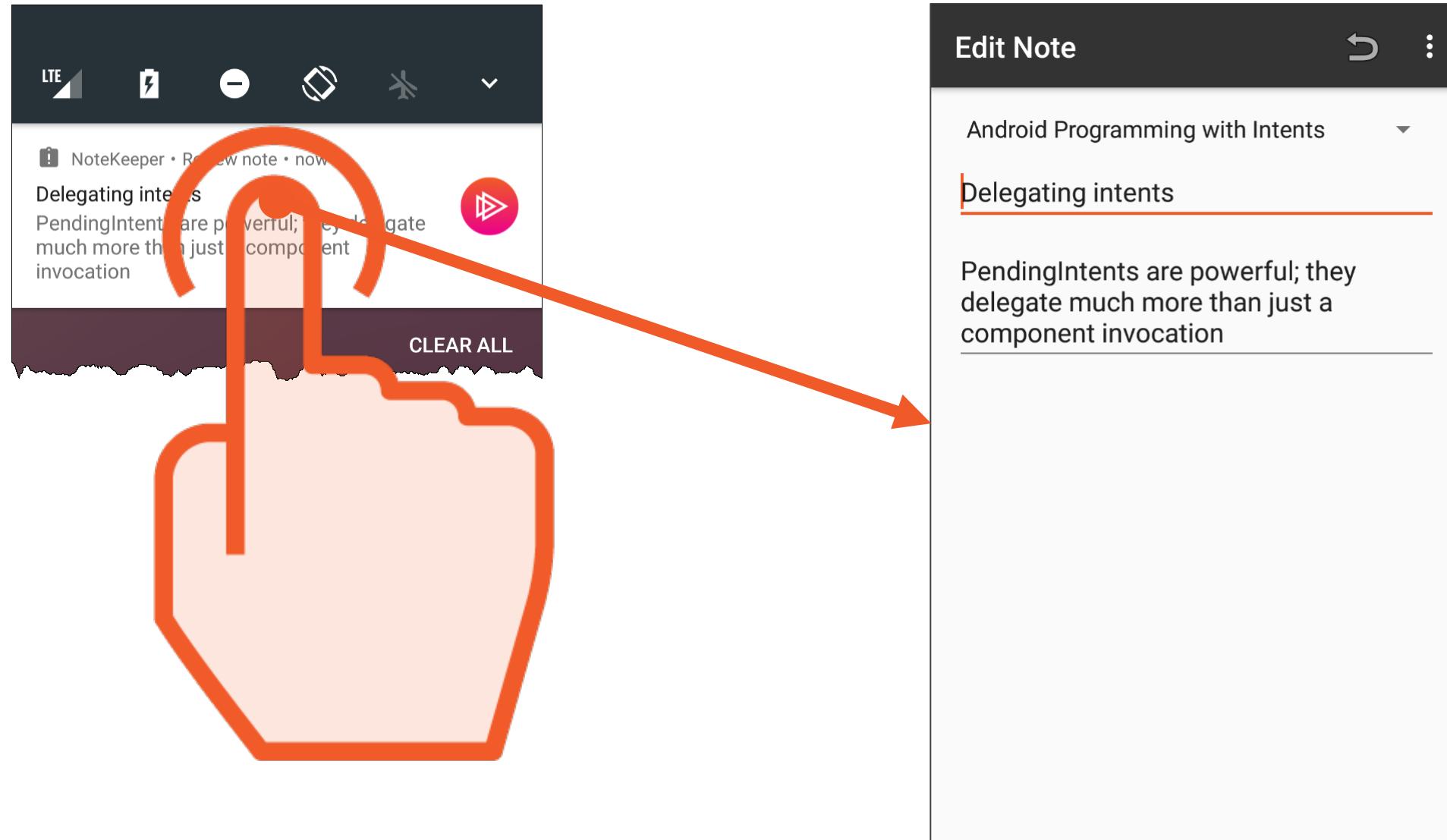
- Provides ability to display a relatively large bitmap

## **InboxStyle**

- Provides ability to display a list of up to five items



# Launching an Activity from a Notification



# Launching an Activity from Within Our App

## **Steps to launch activity from within app**

- Create an intent
- Associate extras with the intent
- Call `startActivity` with intent



# Launching an Activity from a Notification

**Notifications are displayed by the system**

- Notification exists outside of our app

**App must delegate launch instructions**

- Create an intent
- Associate extras with intent
- Wrap up the act of launching an activity with the intent



# Launching an Activity from a Notification

## PendingIntent

- Wraps an intent
- Wraps the action to perform with intent
- Can be passed outside the app

## Creating PendingIntent

- Create intent w/ extras as normal
- Call PendingIntent.getActivity method
- Returns PendingIntent instance that wraps intent and startActivity action



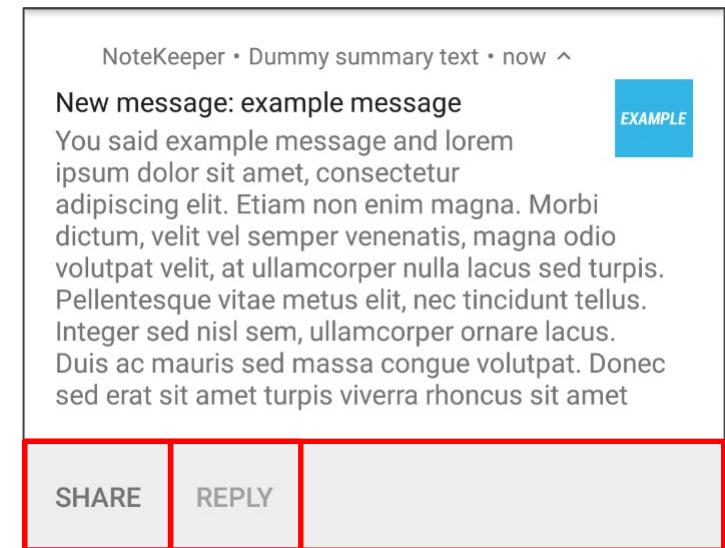
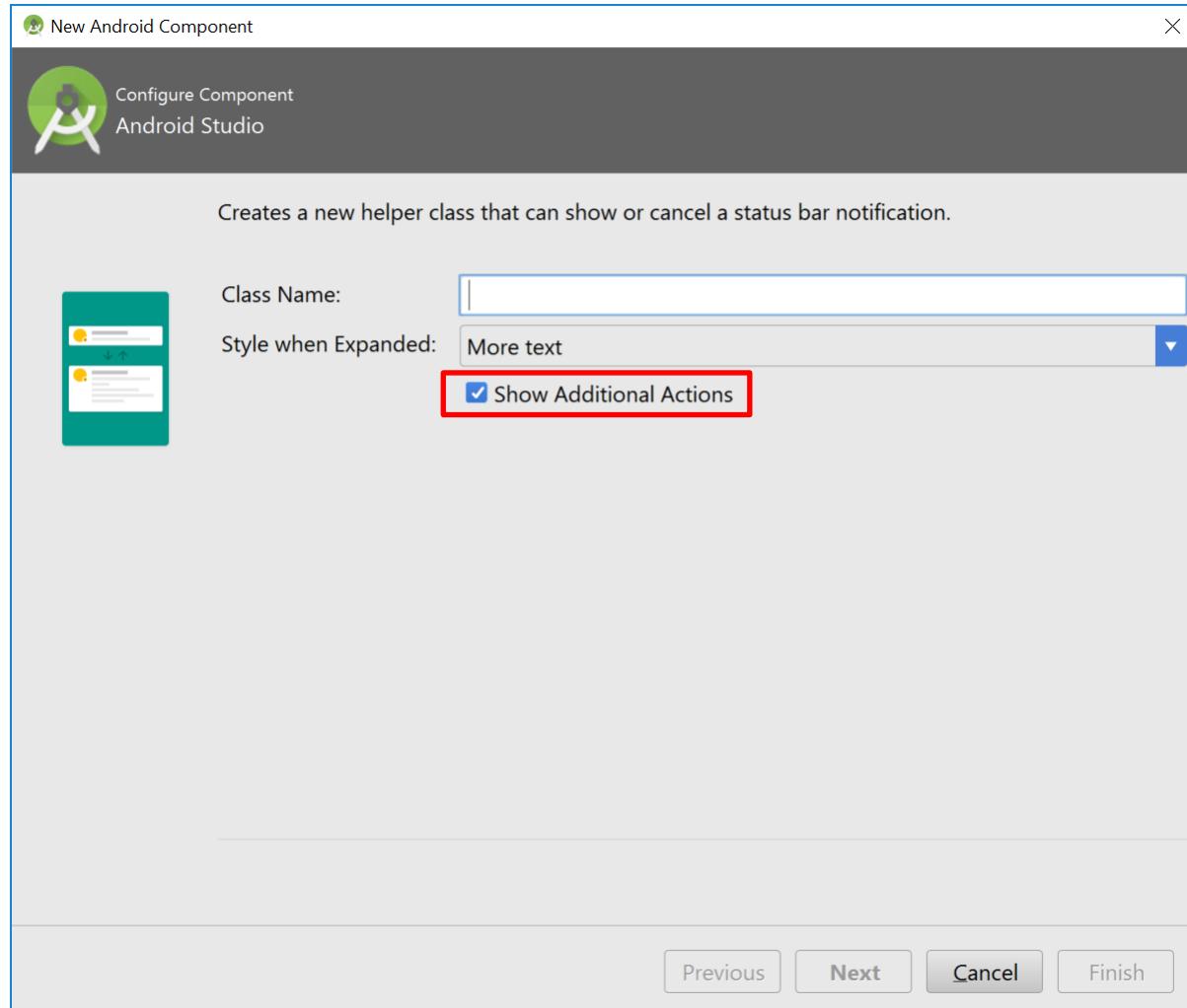
# Launching an Activity from a Notification

## **Associate PendingIntent with notification**

- Use builder setContentIntent method
- Notification will perform PendingIntent action when tapped



# Additional Actions



# Additional Actions

**Actions added w/ builder addAction method**

- Icon
- Display text
- PendingIntent



# Additional Actions

## **Added actions not always visible**

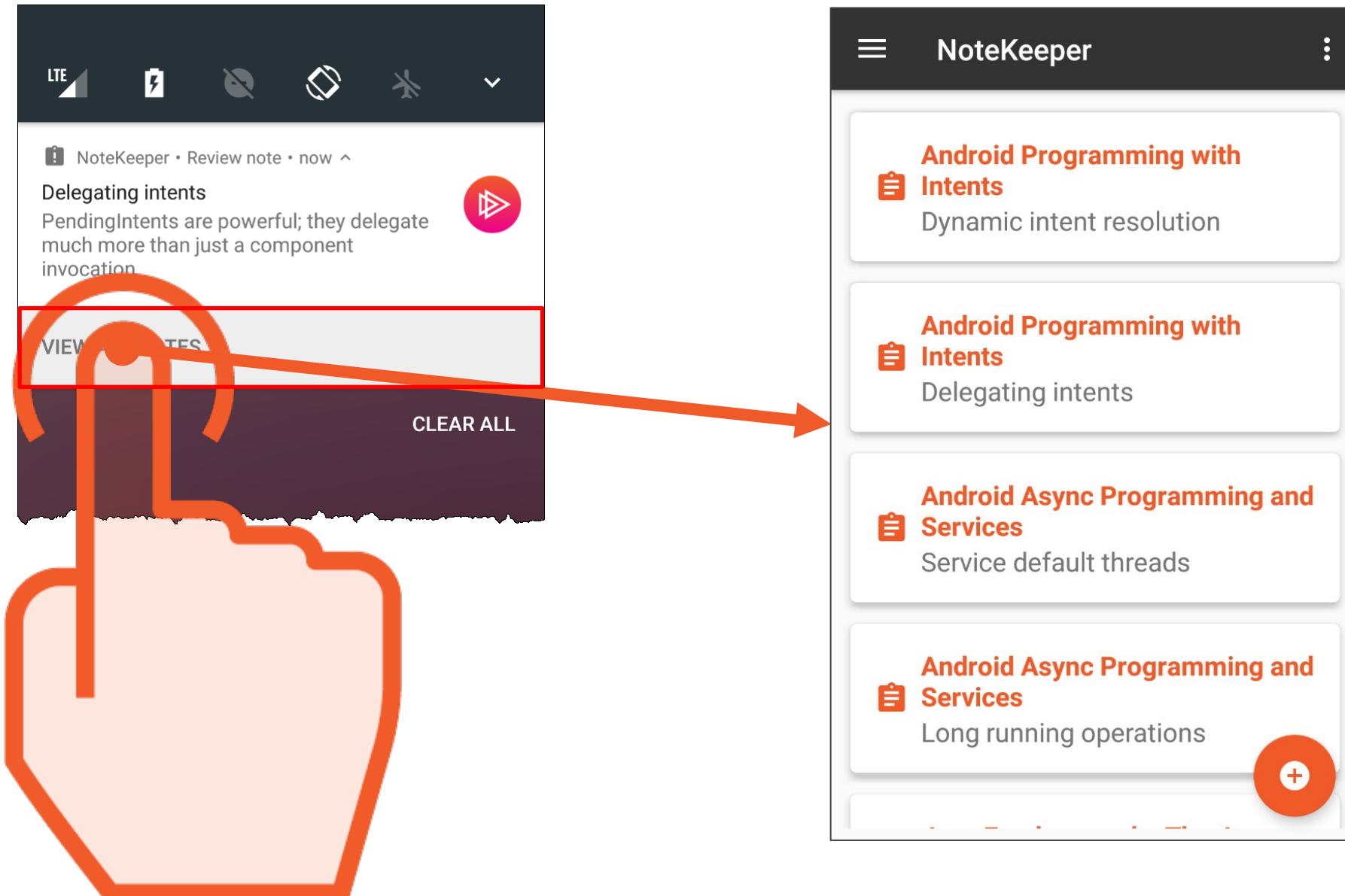
- Visible only when notification expanded

## **Autocancel doesn't apply to added actions**

- App must explicitly call cancel to remove notification



# Additional Actions



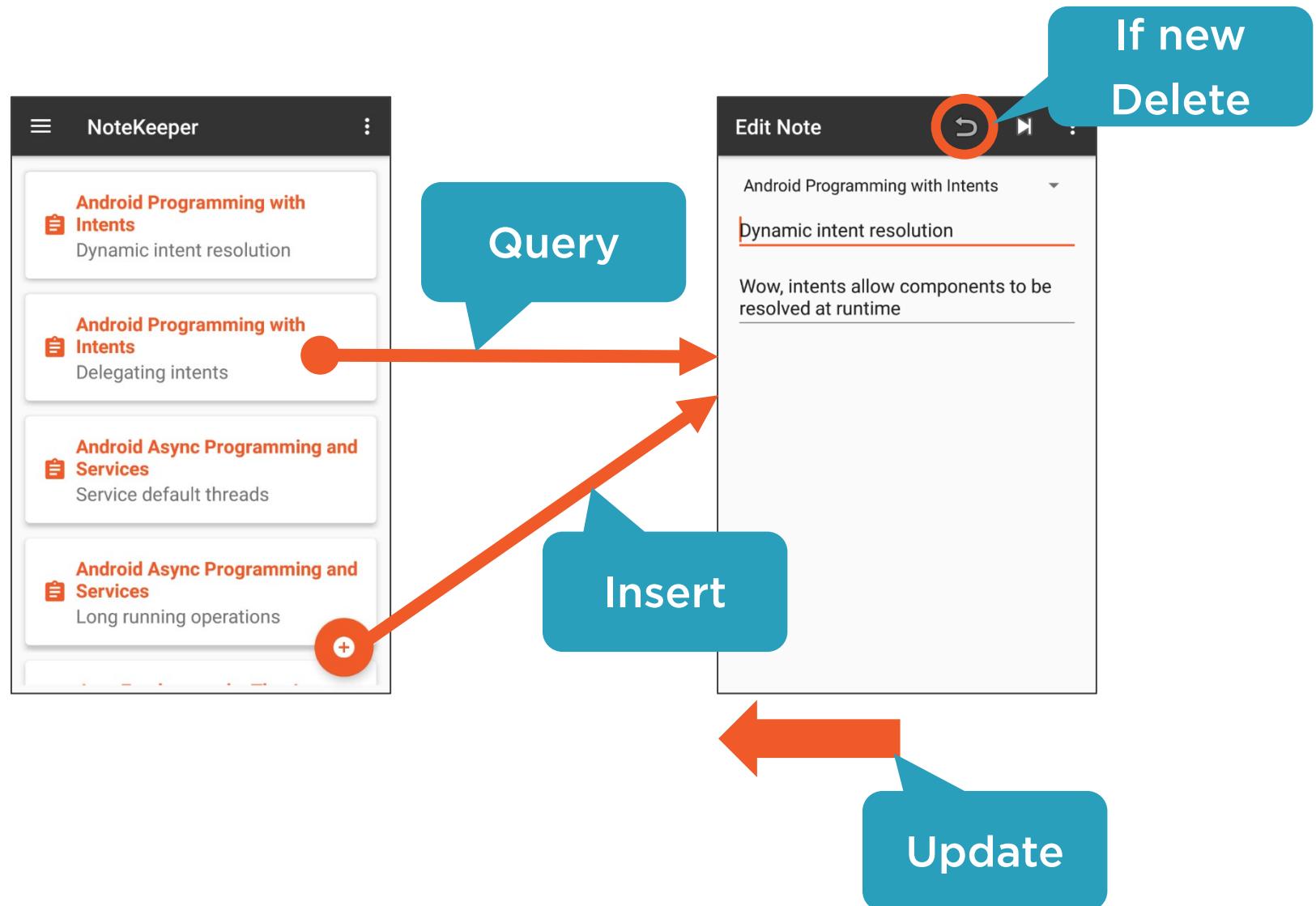
# Creating and Displaying Notifications

## Android Studio generated helper class

- Exposes static notify & cancel methods
- Create & display notification with notify
- Subsequent calls to notify replace previously displayed notification
- Explicitly removes notification



# Note Operations in Our App



# Content Provider Operations

## Support range of data operations

- Query
- Insert
- Update
- Delete

## Accessing a content provider

- CursorLoader commonly used for query
- Need a way to handle non-query



# Accessing a Content Provider



# Accessing a Content Provider



# Content Resolver

## Available from current context

- Use getContentResolver method
- Returns ContentResolver reference

## Serves as operation intermediary

- Exposes methods for each operation
- Methods accept a URI
- Locates content provider
- Delegates operation to content provider



# Inserting a Row into a Table

## Use ContentResolver insert method

- Pass URI of table
- Pass values as ContentValues
- Returns URI of new row



# Inserting a Row into a Table

## Handled by ContentProvider insert method

- Receives URI and ContentValues
- Determine target with UriMatcher.match
- Insert into SQLite database
- Return row URI

## Row URI

- Based on table URI
- Append row ID value as path to end
- Use ContentUris.withAppendedId



# Data Interaction and Row URIs

## Row URI

- Based on table URI
- Row ID appended as path to end

## Primary way to interact with a specific row

- When querying for specific row
- When updating or deleting specific row
- Behaves the same as passing the table URI with “\_id=?” selection criteria



# Data Interaction and Row URIs

## **Content Provider URI Handling**

- Need to handle table URIs
- Need to handle row URIs



# Data Interaction and Row URIs

## UriMatcher and row URIs

- Match Row URI separate from table URI
- Supports wildcard matching for ID value
- Use # in place of ID value in addURI call

`content://com.jwhh.jim.notekeeper.provider/notes/#`



# Data Interaction and Row URIs

## **Content Provider handling of row URIs**

- Database needs “\_id=?” selection criteria
- Extract \_ID value from URI
- Use ContentUris.parseId



# Interacting with Other Applications

## **Content providers accessible by other apps**

- App can interact with data
- App may not know much about the data

## **Can make more information available**

- Make contract class available
- Identify the mime type of the data returned by each URI



# Make Contract Class Available

## **Build jar or Android library**

- Contains only the contract class
- <http://bit.ly/buildandroidlib>

## **Other app references jar/library**

- Allows app to use the constants contained in the contract class



# “OtherApp” Using Content Provider

```
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
    Uri uri =  
        Uri.parse("content://com.jwhh.jim.notekeeper.provider/courses");  
    String[] columns = {  
        "_id",  
        "course_title",  
        "course_id" };  
  
    return new CursorLoader(this, uri, columns,  
        null, null, "course_title");  
}
```



# “OtherApp” Using Content Provider

```
#import com.jwhh.jim.notekeeper.NoteKeeperProviderContract.Courses;  
  
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
  
    Uri uri = Courses.CONTENT_URI;  
  
    String[] columns = {  
        "_id",  
        "course_title",  
        "course_id" };  
  
    return new CursorLoader(this, uri, columns,  
        null, null, "course_title");  
}
```



# “OtherApp” Using Content Provider

```
#import com.jwhh.jim.notekeeper.NoteKeeperProviderContract.Courses;  
  
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
  
    Uri uri = Courses.CONTENT_URI;  
  
    String[] columns = {  
        Courses._ID,  
        Courses.COLUMN.Course_TITLE,  
        Courses.COLUMN.Course_ID };  
  
    return new CursorLoader(this, uri, columns,  
        null, null, Courses.COLUMN.Course_TITLE);  
}
```



# Identify Each URIs Mime Type

**Mime types describe data**

- Many common mime types exist

**Our data doesn't fit common mime types**

- Cursor based data
- Data is application defined
- We need to construct or own mime type



Identify Each  
URIs Mime Type

## Constructing our mime type

- Table name
- Content provider authority
- Identify as vendor mime type
- Whether returns one or multiple rows



# Identify Each URLs Mime Type

vnd.android.cursor.item/

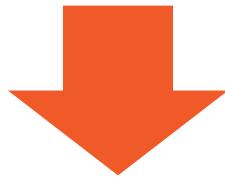
vnd.com.jwhh.jim.notekeeper.provider.courses

vnd.android.cursor.dir/



# Identify Each URLs Mime Type

content://com.jwhh.jim.notekeeper.provider/courses



vnd.android.cursor.dir/vnd.com.jwhh.jim.notekeeper.provider.courses



# Identify Each URLs Mime Type

content://com.jwhh.jim.notekeeper.provider/courses/8



vnd.android.cursor.item/vnd.com.jwhh.jim.notekeeper.provider.courses



# Identify Each URIs Mime Type

## Implement `getType` method

- Return appropriate mime type
- Build with simple string concatenation

## ContentResolver helpful constants

- `CURSOR_DIR_BASE_TYPE`
- `CURSOR_ITEM_BASE_TYPE`



# Make Contract Class Available

## **Build jar or Android library**

- Contains only the contract class
- <http://bit.ly/buildandroidlib>

## **Other app references jar/library**

- Allows app to use the the content provider constants



```
#import com.jwhh.jim.notekeeper.NoteKeeperProviderContract.Courses;  
  
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
  
    Uri uri = Uri.parse(Courses.CONTENT_URI);  
  
    String[] columns = {      "_id",  
        "course_title",  
        "course_id"};  
  
    return new CursorLoader(this, uri, columns,  
        null, null, "course_title");  
  
}
```



# Make Constants Available

## Make contract class available

- Build jar or Android library
- Contains only the contract class
- Other app references jar/library

## Building a jar or Android library

- <http://bit.ly/buildandroidlib>



# Content Provider Organization

## **Content providers expose tables**

- Normally expose multiple tables
- Often correspond to database tables
- May also provide an abstraction

## **Need a way to describe content provider**

- Identify available tables
- Identify table columns



# Content Provider Contract Class

## Presents the public appearance

- Avoid exposing details of structure
- Focus is on data access
- Includes base URI for content provider

`content://com.jwhh.jim.notekeeper.provider`



# Content Provider Contract Class

**Table information is in nested classes**

- Separate class for each table
- Content provider's public representation

**Each class describes how to access a table**

- Names of available columns
- Table URI



# Content Provider Contract Class

## Table URIs

- Starts with content scheme
- Includes provider authority
- Table is added as a path
- Use Uri.withAppendedPath static method

`content://com.jwhh.jim.notekeeper.provider/courses`



# Content Provider Contract Class

**Each table class describes table access**

- Table URI
- Names of available columns

**Defining column name constants**

- Could add constants directly to classes



# Content Provider Contract Class

## Columns often duplicated across tables

- Same name and meaning
- Could lead to redundant constants
- Better to manage column constants separately from table classes



# Content Provider Contract Class

## Column constants managed in interfaces

- Interfaces group related columns
- Simplifies organization and maintenance

## Interfaces nested within contract class

- Not for use outside of provider
- Usually marked protected

## Associating constants with table classes

- Classes implement appropriate interfaces



# Matching Content URIs

## **Content provider URI handling**

- Often need to support many URIs
- Interpreting URIs can be challenging
- URI patterns can become complex



# Matching Content URIs

## **UriMatcher**

- Handles details of interpreting URIs
- Translates a URI to an integer constant



# Matching Content URIs

## Preparing UriMatcher in a content provider

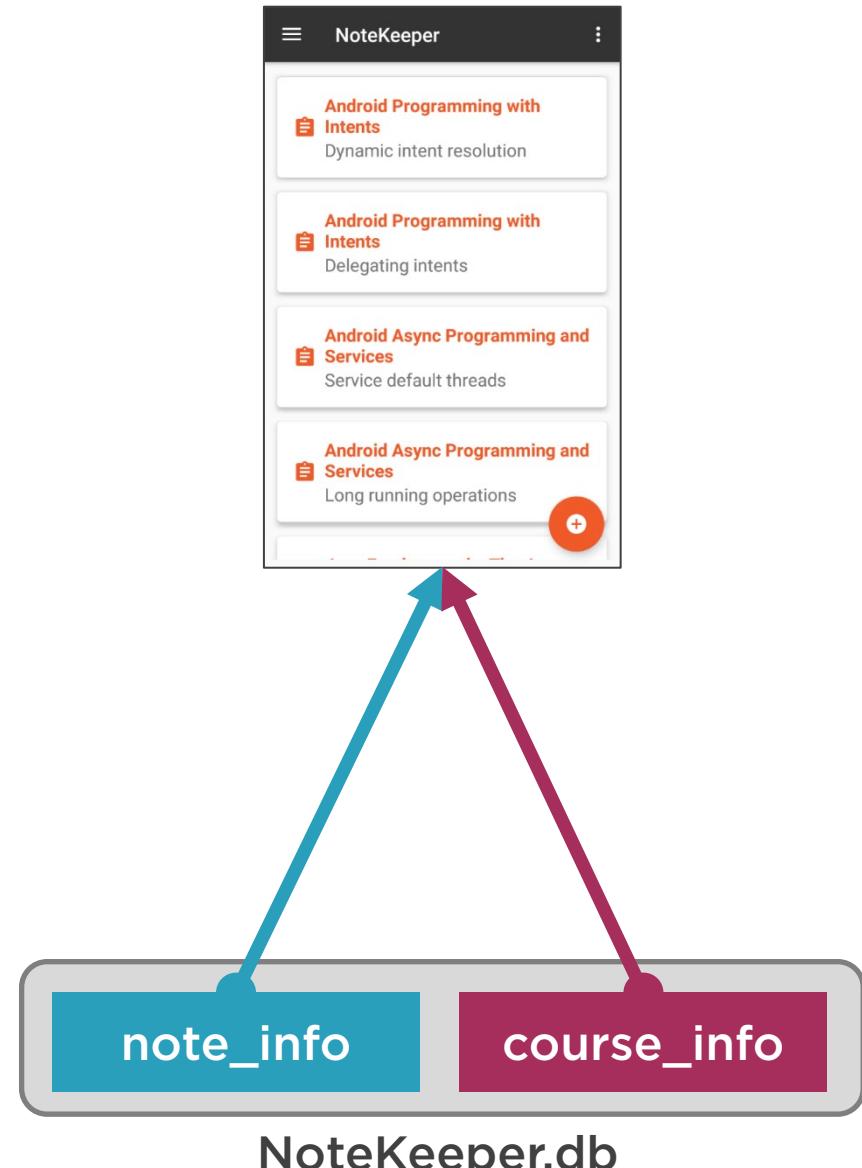
- Usually have a static member field
- Add valid URIs in a static initializer
- Use addURI method
- Map each URI to an integer constant

## Handling requests with UriMatcher

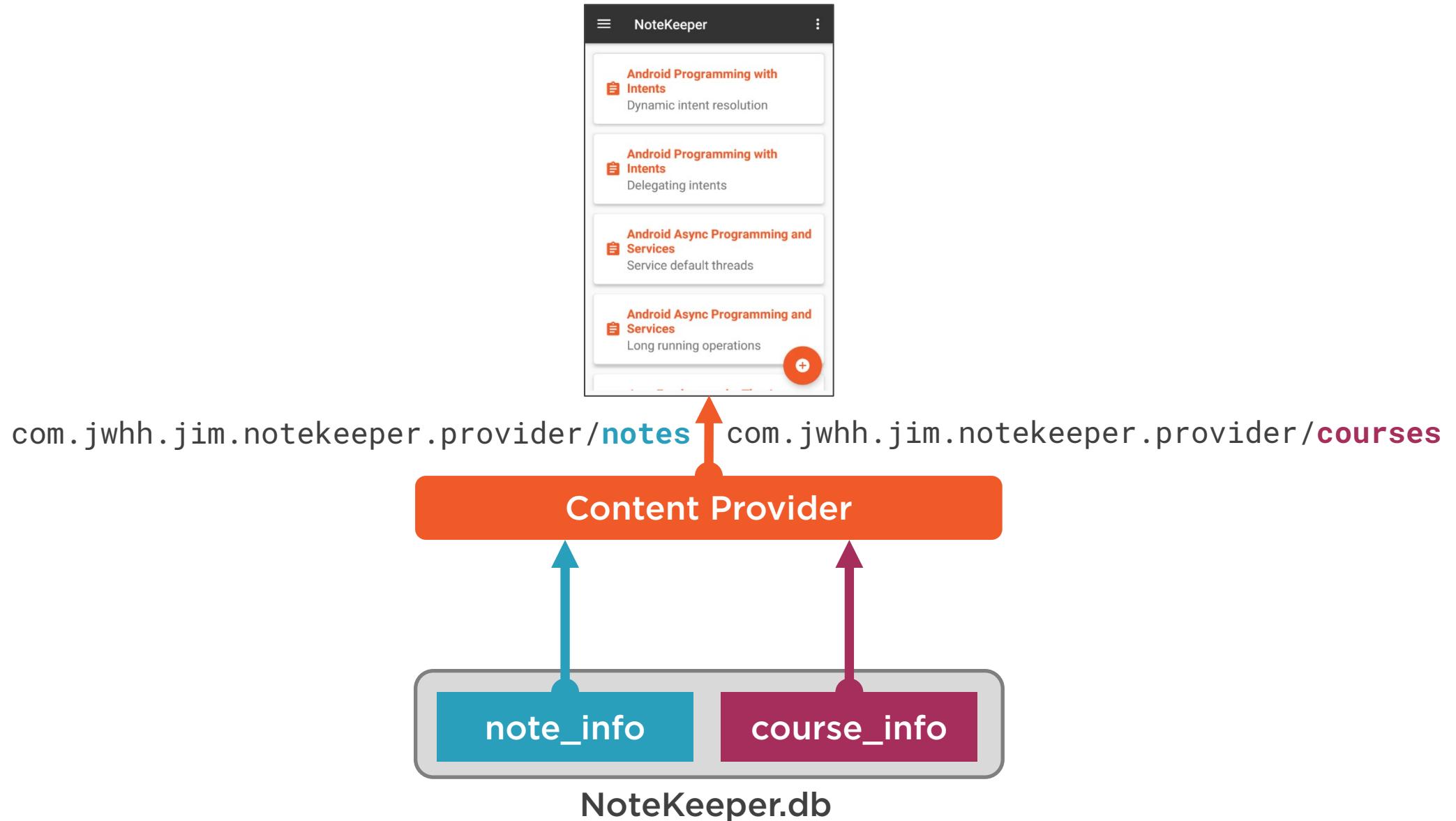
- Use match method
- Translate URI to an integer constant
- Take action based on constant



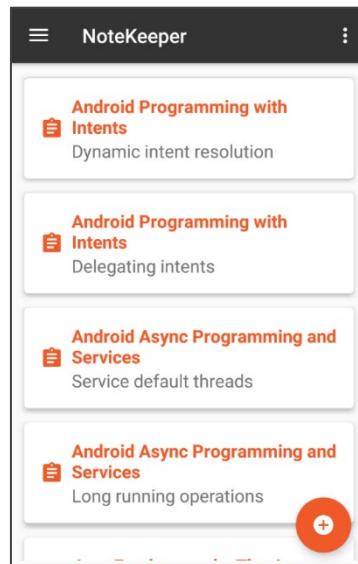
# Table Abstraction



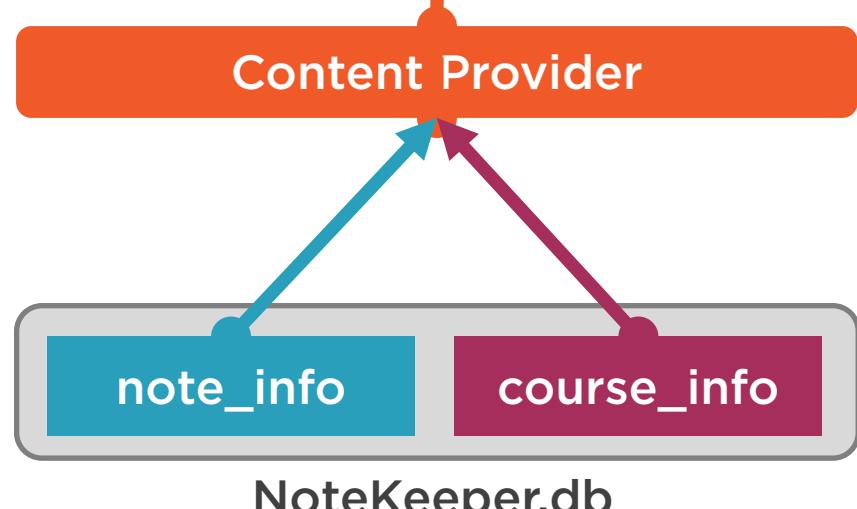
# Table Abstraction



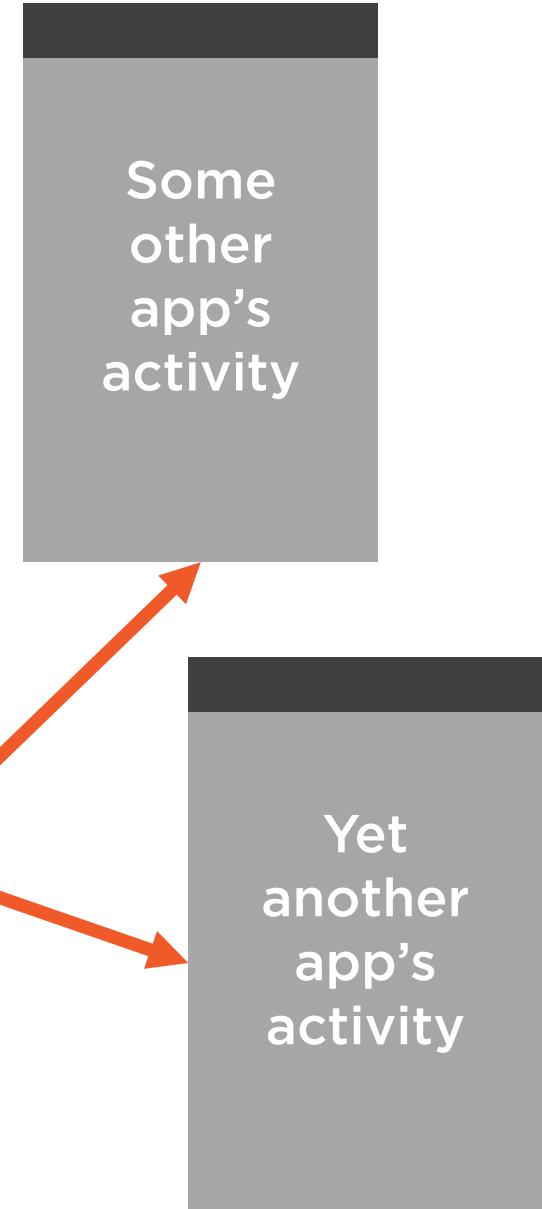
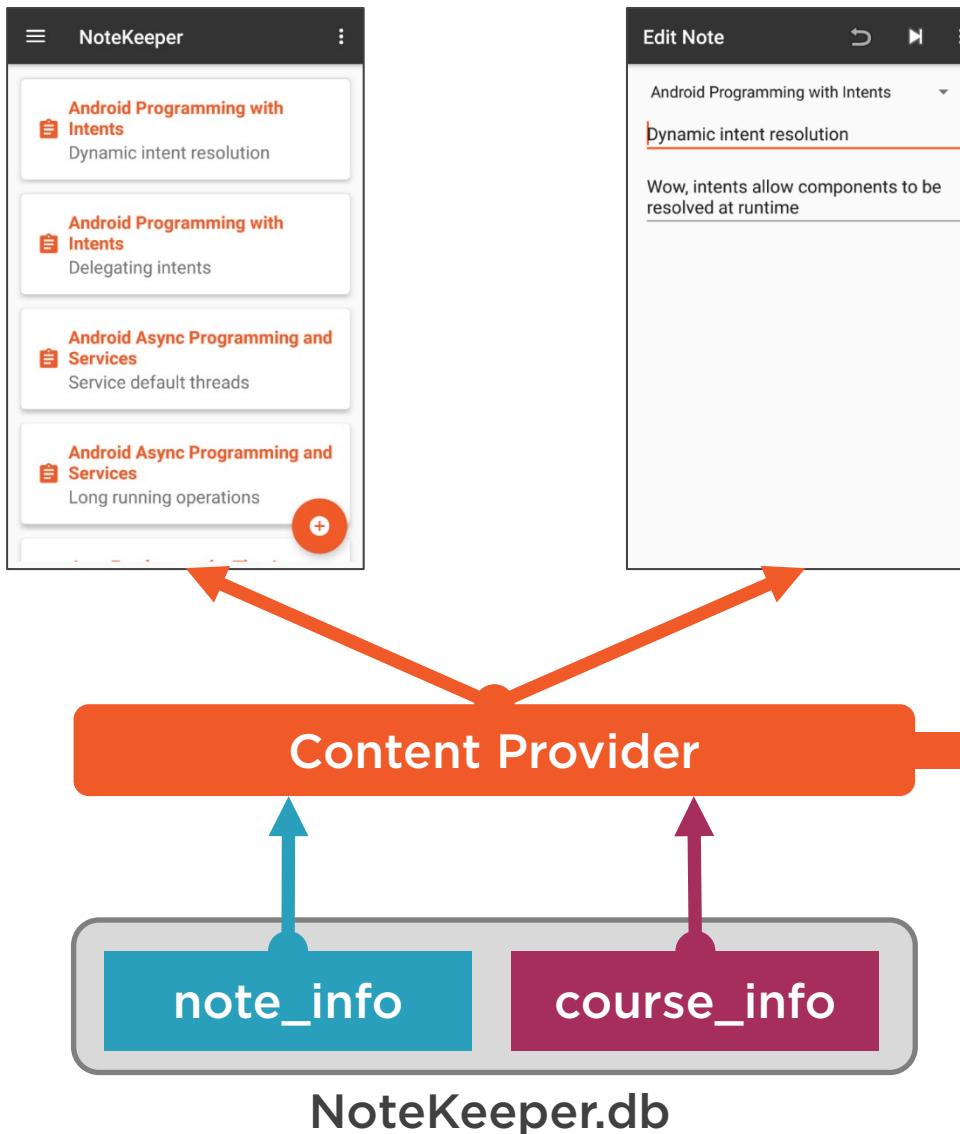
# Table Abstraction



com.jwhh.jim.notekeeper.provider/**notes\_expanded**



# Accessing Data



# Joining Tables

```
query ("note_info JOIN course_info  
      ON note_info.course_id = course_info.courseId", ...);
```



note_title	course_title
Dynamic intent resolution	Android Programming with Intents
Anonymous classes	Java Fundamentals: The Java Language
Delegating intents	Android Programming with Intents



# Content Provider Contract Class

**Holds information about accessing data**

- Includes base URI for content provider
- Avoid exposing details of structure
- Present the public appearance

**Content provider organization**

- Content available as data entities
- Entities contain data values



# Content Provider Contract Class

## Available data values in nested interfaces

- Values often tied to table columns
- Interfaces usually marked protected

## Interfaces group related columns

- Simplifies organization and maintenance



# Content Provider Contract Class

## **Data entities in nested public classes**

- Often tied to tables
- May be an abstraction of tables

## **Classes describe values and access**

- Implements interfaces for exposed values
- Includes URI for accessing entity



# Content Provider Contract Class

## Content entities accessed with URIs

- Starts with content scheme
- Includes provider authority
- Specific content is added as a path
- Construct with Uri.withAppendedPath

`content://com.jwhh.jim.notekeeper.provider/courses`



# Content Provider

An Android component that encapsulates data, exposes that data through a standard interface, and optionally makes that data available to multiple programs.



# Content Provider

## Content provider

- A way to expose data
- Data exposed through a standard API

## Content provider visibility

- Can be limited to app that implements it
- Can be available to other apps



# Content Provider

**Concept is independent of data storage**

- Local and/or remote data
- Static and/or dynamically-produced data
- Implementation encapsulates details



# SQLite vs. Content Provider

<b>SQLite</b>	<b>Content Provider</b>
<b>Data storage and management solution</b>	<b>Data access solution</b>
<b>Data accessed through SQLite library</b>	<b>Data accessed through standard API</b>
<b>Accessible by app that owns file</b>	<b>Access can be available to other apps</b>



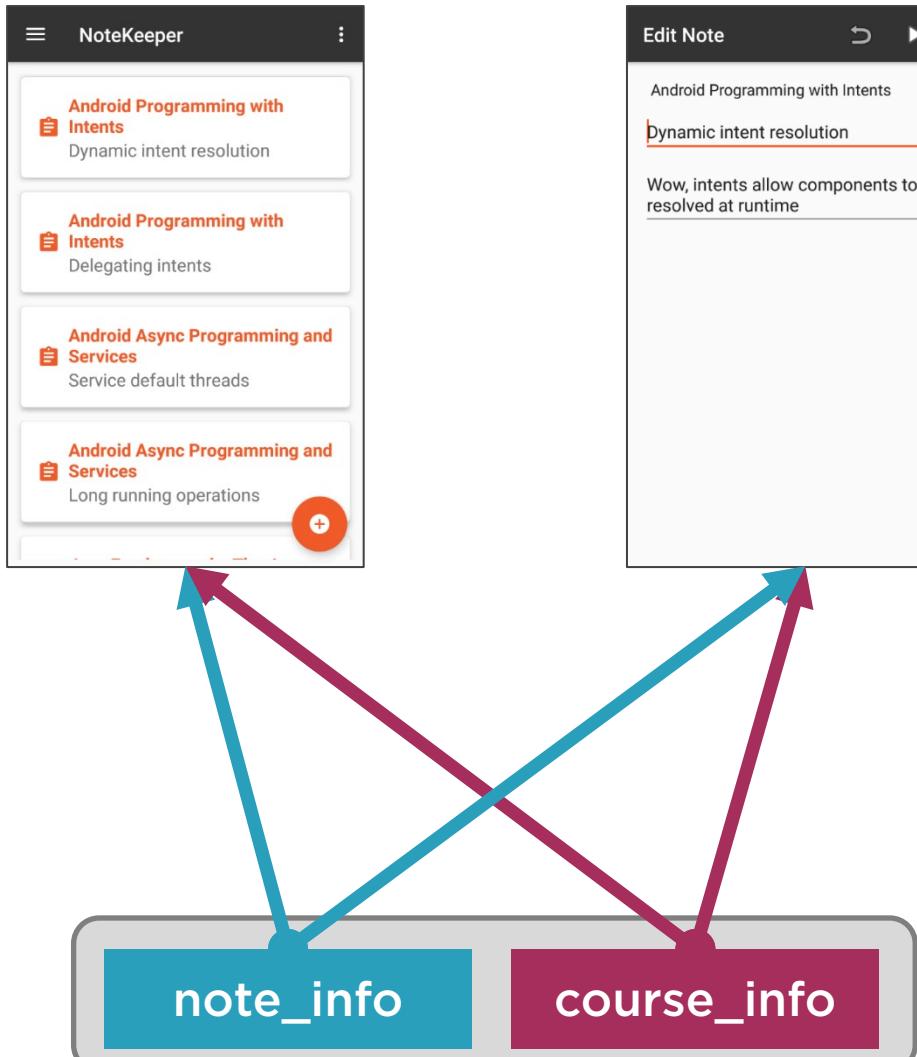
# Content Provider Common Implementation

## **Commonly used in conjunction with SQLite**

- SQLite provides the backing store
- Content provider encapsulates details
- Content provider may make data available to other apps



# Accessing Data

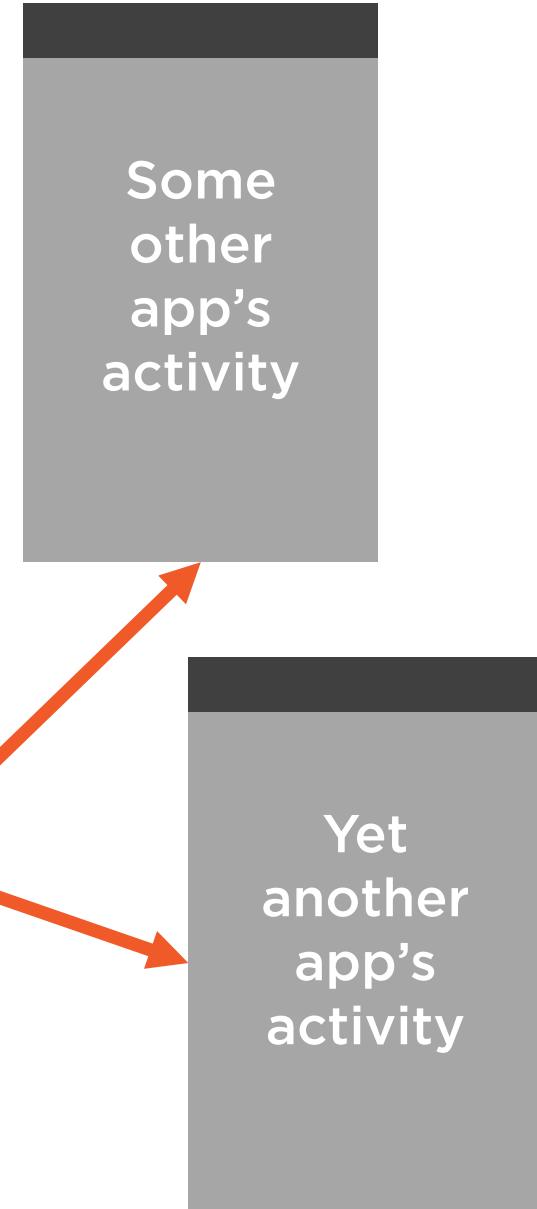
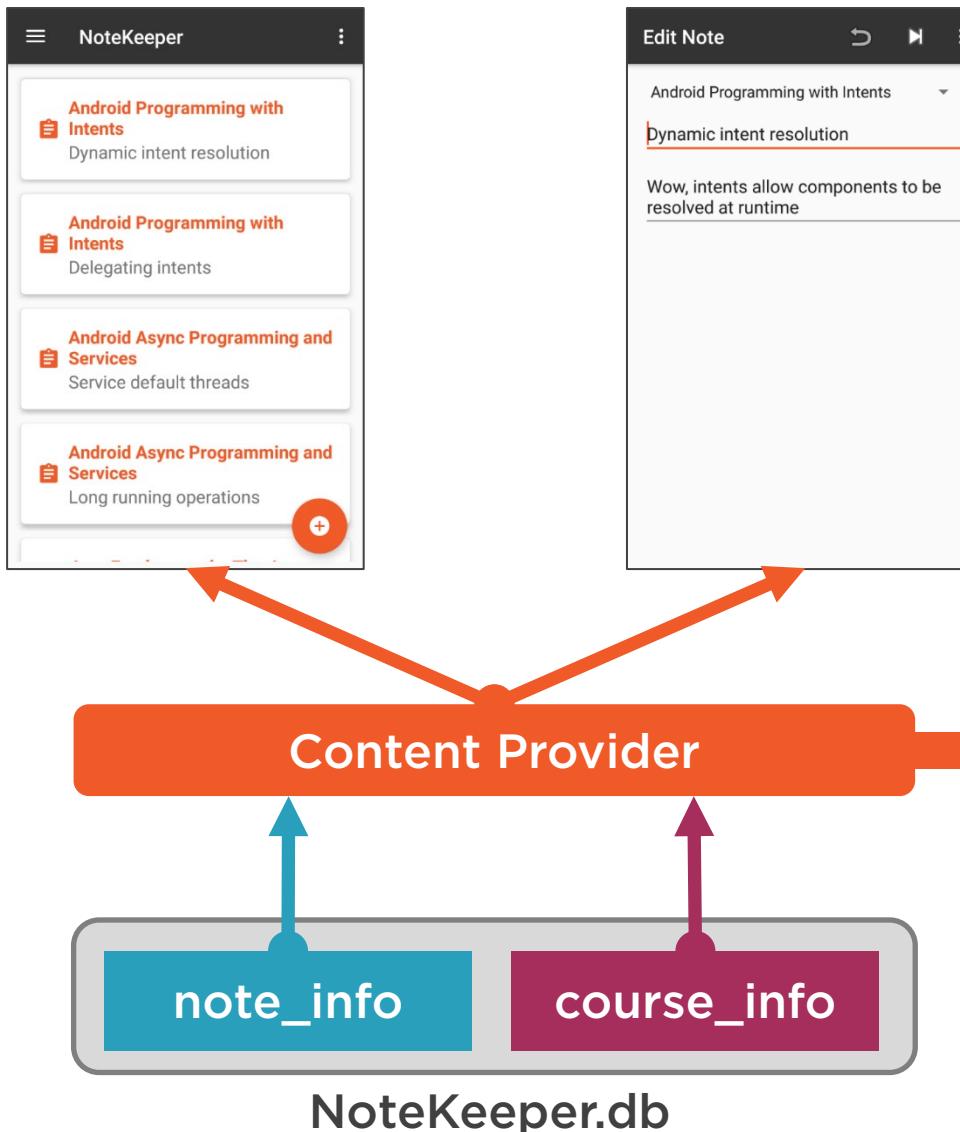


Some  
other  
app's  
activity

Yet  
another  
app's  
activity



# Accessing Data



# Creating a Content Provider

## **Extend ContentProvider class**

- Implement lifecycle methods
- Implement data lookup method
- Implement data modification methods



# Creating a Content Provider

## Content provider identification

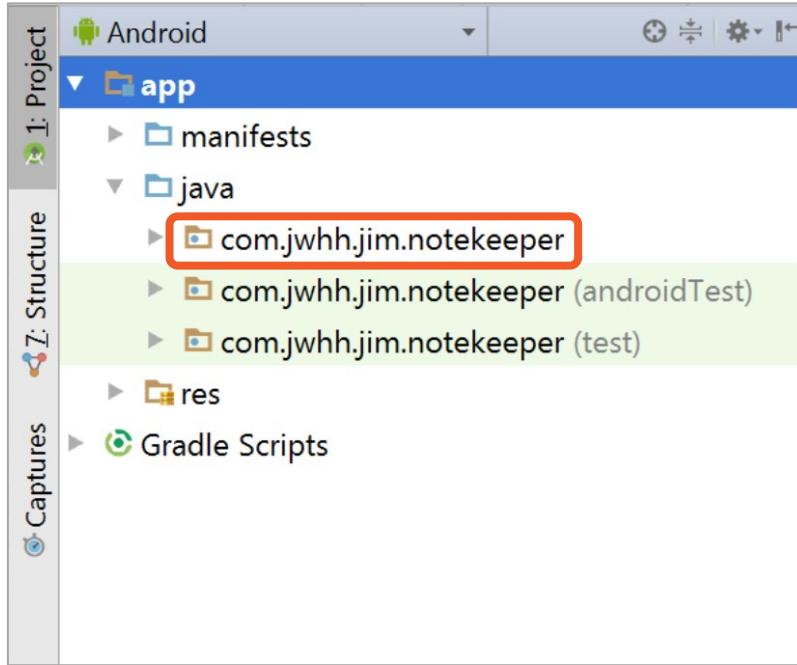
- May be visible throughout device
- Identification should be globally unique

## Associate an authority

- Uniquely identifies content provider
- Use reverse domain name format
- Normally use app package name followed by *.provider*



# Content Provider Authority



com.jwhh.jim.notekeeper



# Creating a Content Provider

## **Decide visibility to other apps**

- Be explicit in this decision
- Mark as exported to make available



# Implementing a Content Provider Over SQLite

## Encapsulates database related details

- Responsible to connect to database
- Performs all database interactions

## Exposes methods similar to SQLite API

- Data requests handled with query method
- Delegates work to SQLite



# Requesting Data from a Content Provider

## Similar to directly querying SQLite

- Avoid performing queries on main thread

## CursorLoader

- Runs query on background thread
- Cooperates with activity lifecycle

## LoaderManager

- Coordinates loader and activity



# Requesting Data from a Content Provider

## CursorLoader

- Understands querying content providers
- No need to overload loadInBackground
- Pass content provider identifier to CursorLoader constructor



# Requesting Data from a Content Provider

## Identifying content providers

- Use universal resource identifier (URI)
- Has a URI scheme of content
- Includes content provider authority

`content://com.jwhh.jim.notekeeper.provider`



# Requesting Data from a Content Provider

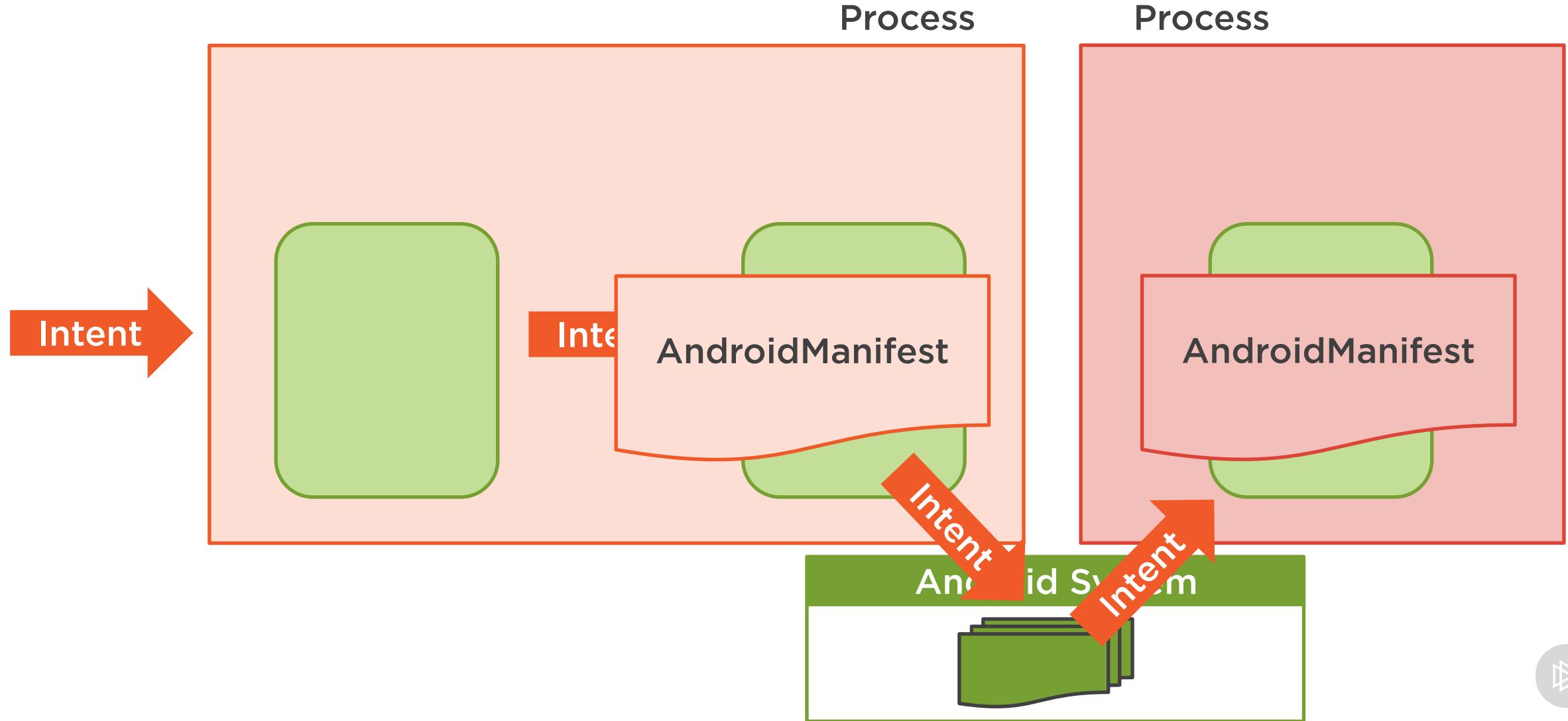
## Identifying content providers

- Use universal resource identifier
- Has a URI scheme of content
- Includes content provider authority

`content://`



# Implicit Intents



# Creating a Content Provider

## Associate an authority

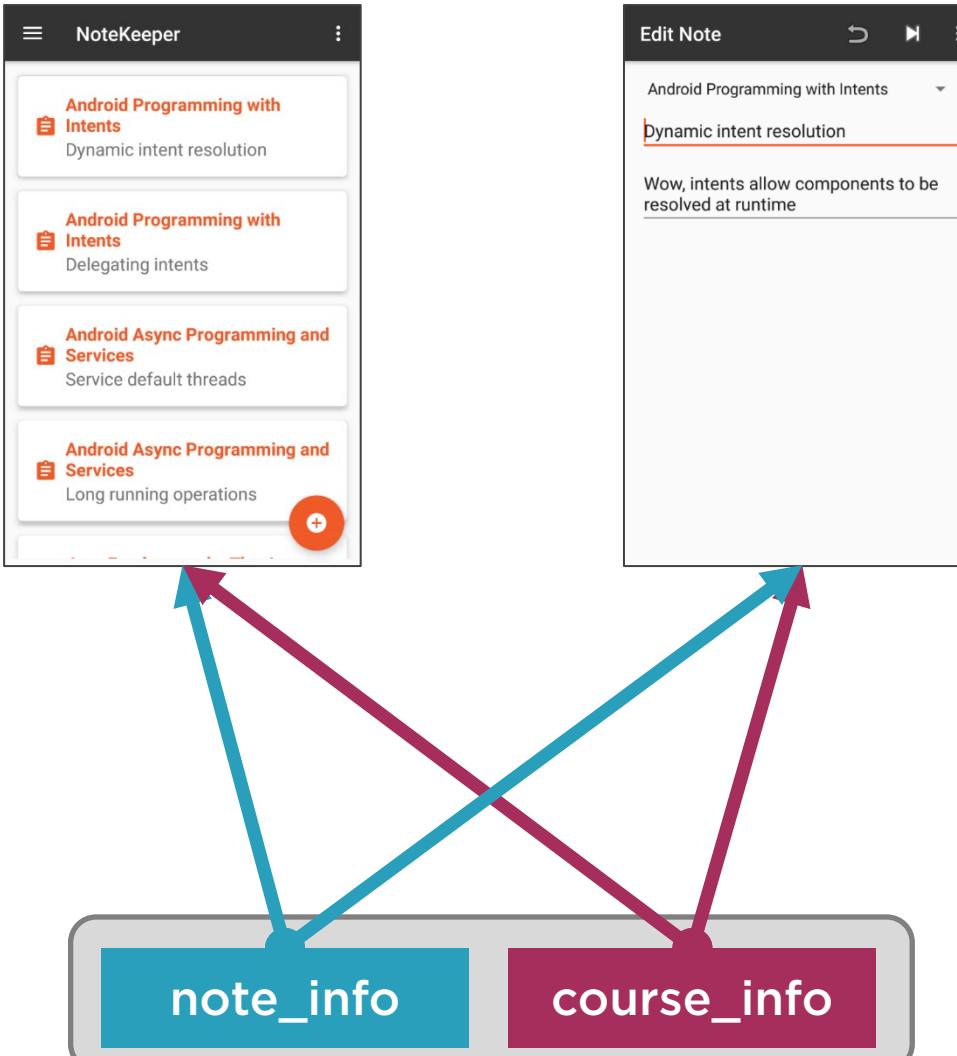
- Uniquely identifies content provider
- Used by Android to locate the content provider

## Specifying the authority

- Use reverse domain naming
- Normally use app package name followed by *.provider*



# Slide 3

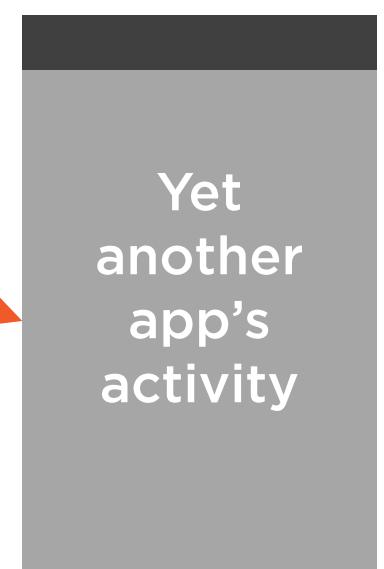
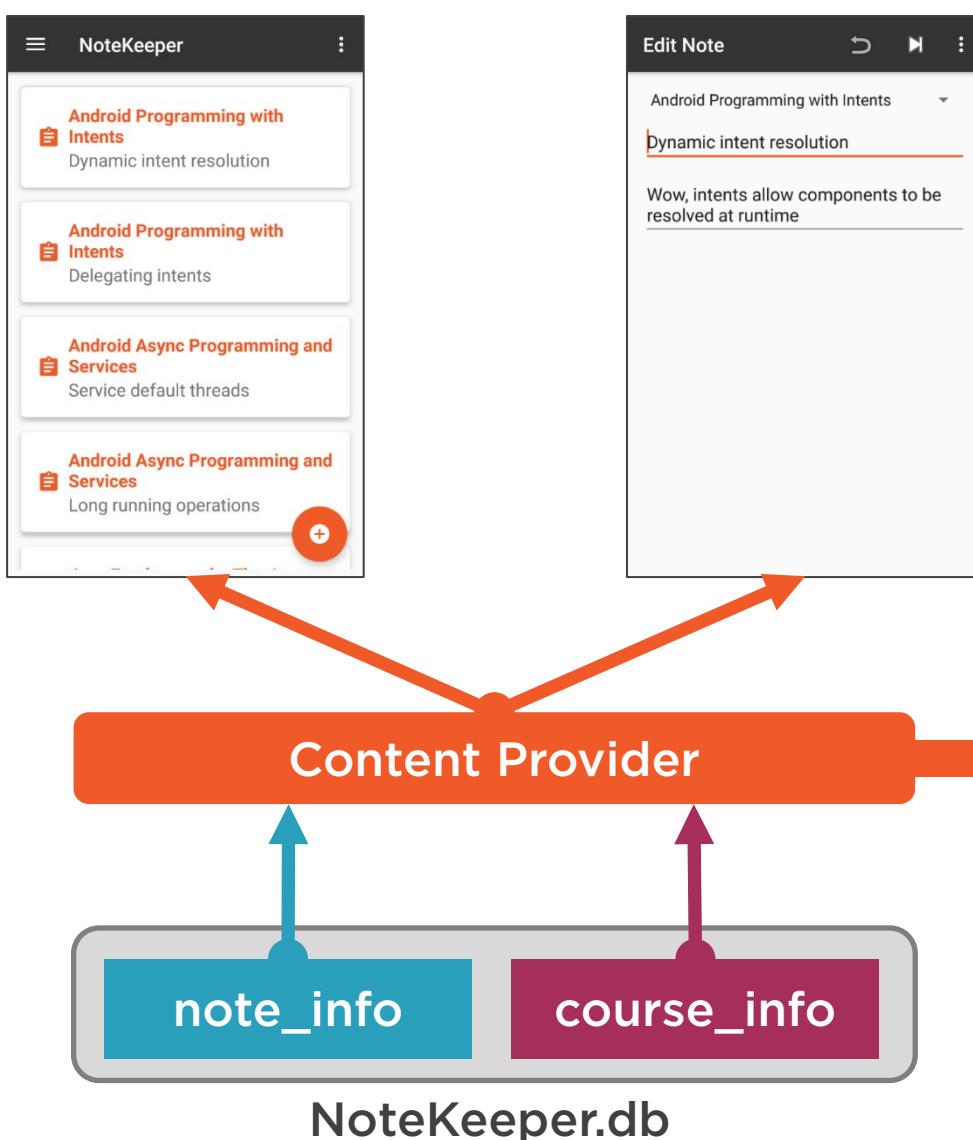


Some  
other  
app's  
activity

Yet  
another  
app's  
activity

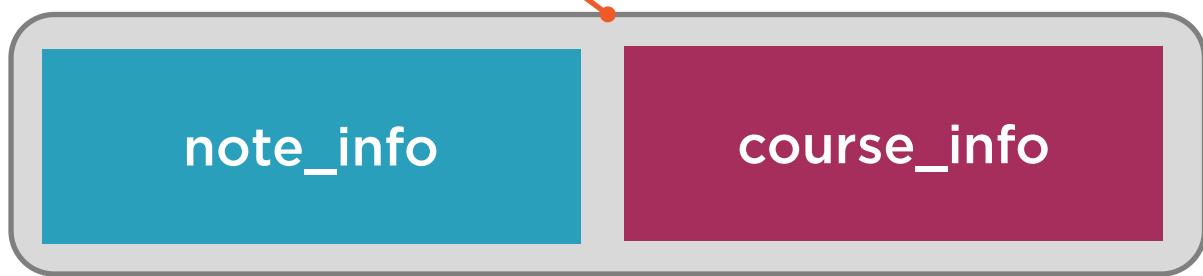


# Slide 4



# Slide 1

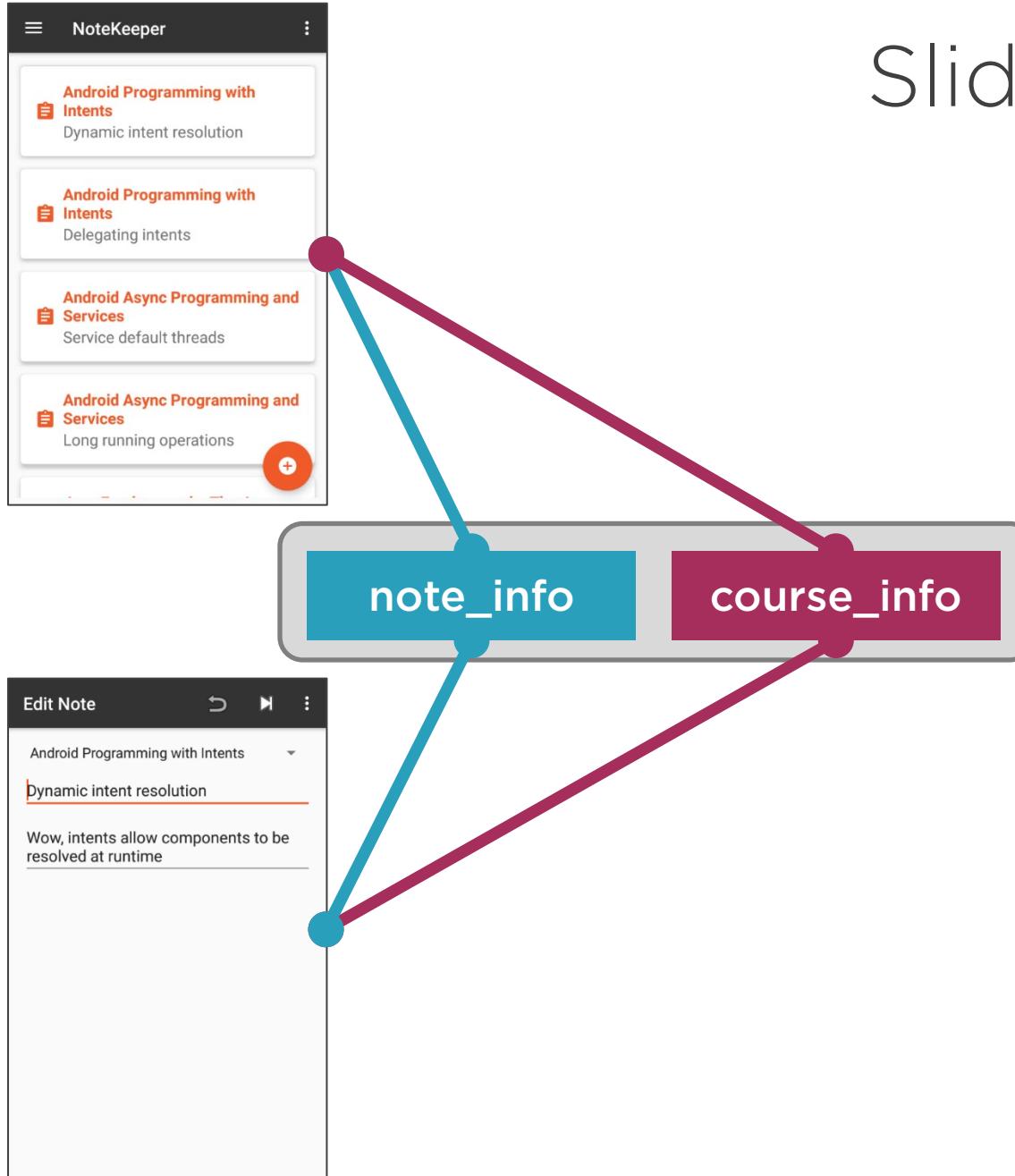
MainActivity



NoteKeeper.db



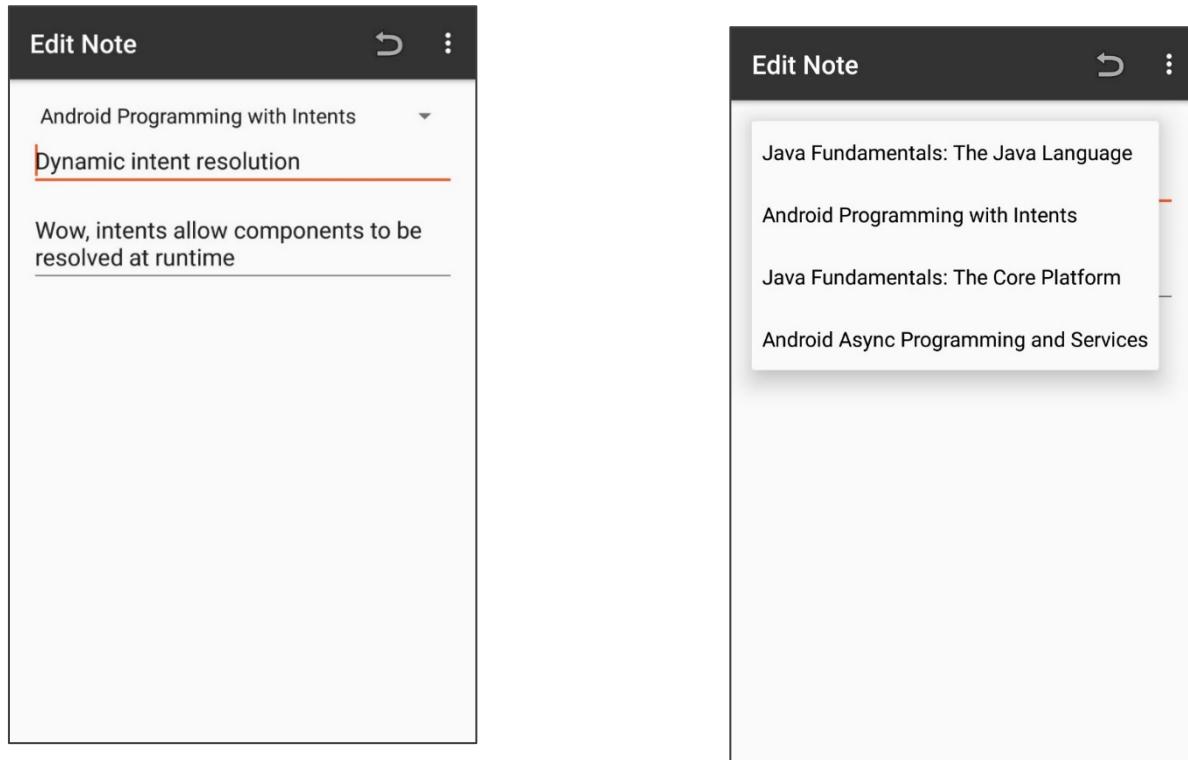
# Slide 2



NoteKeeper.db



# Note Display in Our App



NoteKeeper.db



# Making Data Changes

## Requires connection to database

- Use open helper's getWritableDatabase
- Returns SQLiteDatabase reference

## Interaction is still table based

- Operations performed against a table
- Affects rows & columns within the table



# Making Data Changes

## Update

- Modify column values of existing row(s) in a table

## Insert

- Create a new row in a table

## Delete

- Remove existing row(s) from a table



# Updating Data

## Update

- Modify column values of existing row(s) in a table

## Use **SQLiteDatabase update method**

- Provide table name
- Names of columns to change
- New column values
- Row selection criteria



# Updating Data

note_info			
_id	note_title	note_text	course_id
1	Intent note	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Intent note	PendingIntents are powerful...	android_intents

```
update(note_info, note_title: Intent note, course_id = ?, android_intents)
```



2



# Updating Data

note_info			
_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	My new title	Anonymous classes simplify...	android_async
3	Delegating intents	PendingIntents are powerful...	android_intents

```
update(note_info, note_title: My new title , _id = ?, 2)
```

```
course_id: android_async
```



1



# Updating Data

## Specfying table

- Pass table name

## Specifying selection criteria

- Pass selection clause and arguments

## Specifying columns and values

- Use ContentValues class
- Holds list of column names & values
- Add each name & value with put method



# Inserting Data

## Insert

- Create a new row in a table

## Use `SQLiteDatabase insert` method

- Table name
- ContentValues with column values



# Inserting Data

note\_info

_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Delegating intents	PendingIntents are powerful...	android_intents

insert(note\_info, note\_title: My new note )  
note\_text: I love Android  
course\_id: android\_async



4



# Deleting Data

## Delete

- Remove existing row(s) from a table

## Use `SQLiteDatabase delete` method

- Provide table name
- Row selection criteria



# Deleting Data

note\_info

_id	note_title	note_text	course_id
2	Anonymous classes	Anonymous classes simplify...	java_lang
4	My new note	I love Android	android_async
5	Long running operations	Foreground services can be tie...	android_async

```
delete( note_info, course_id = ?, android_intents )
```



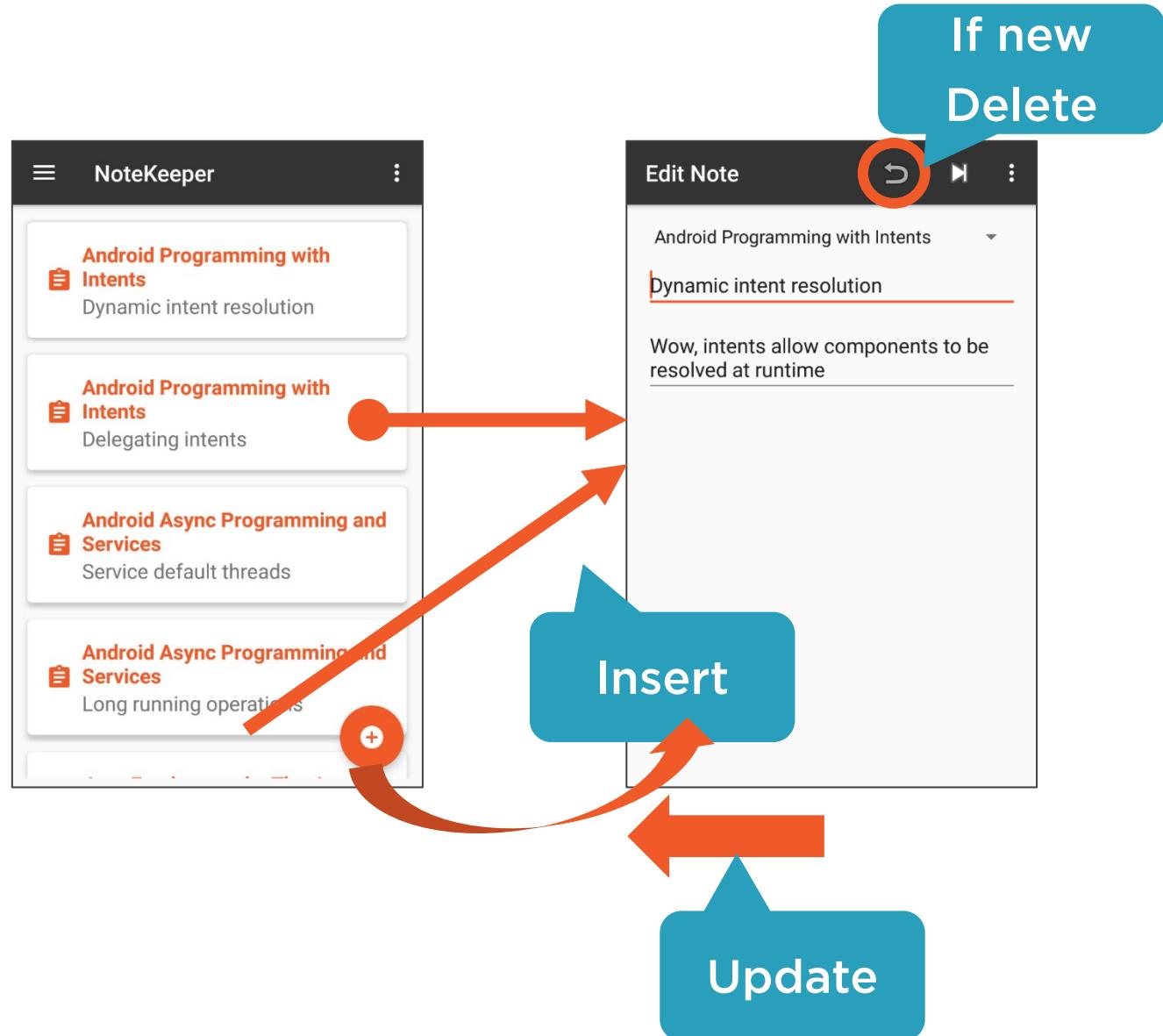
2



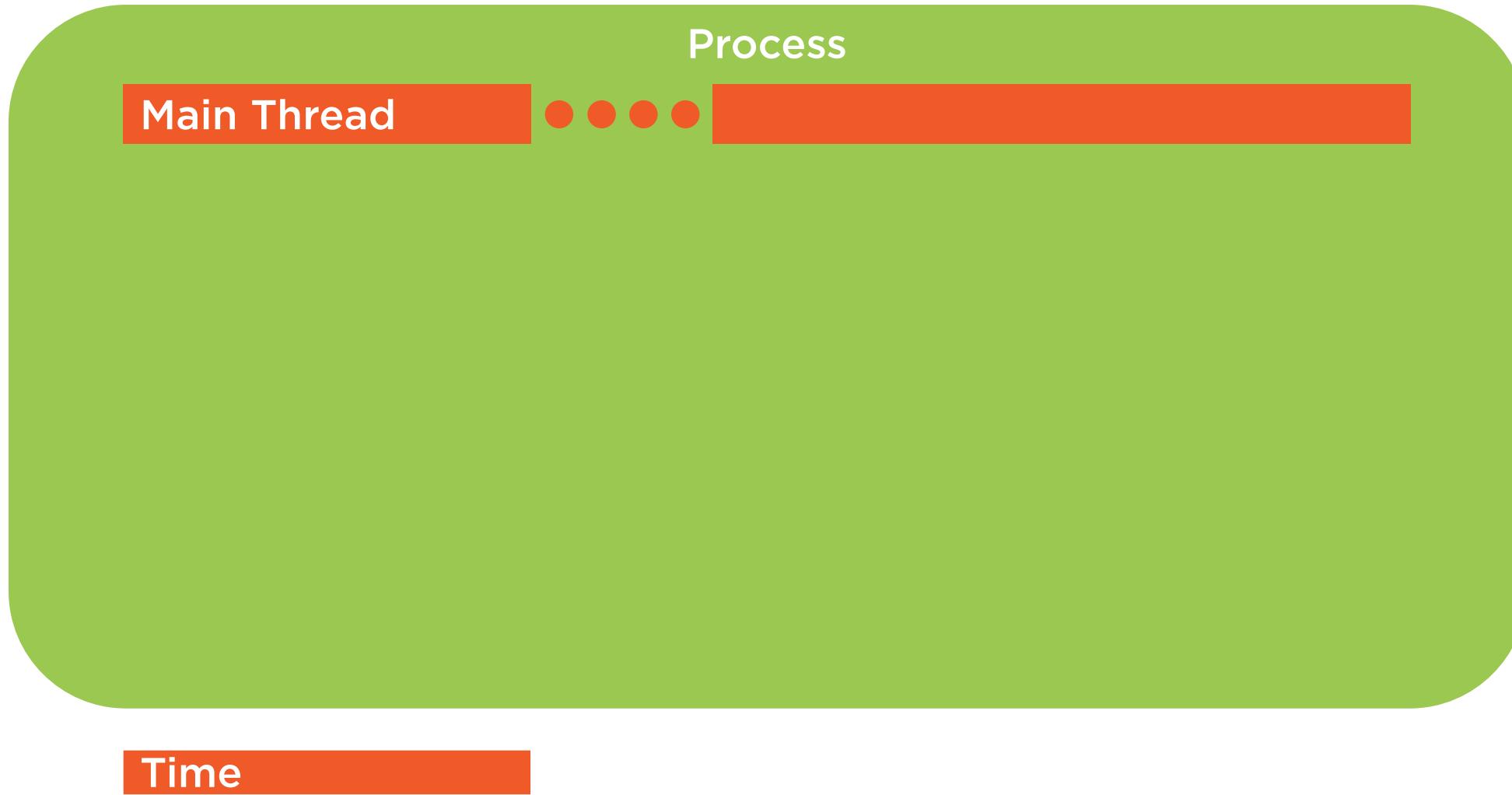
# Note Database Operations in Our App



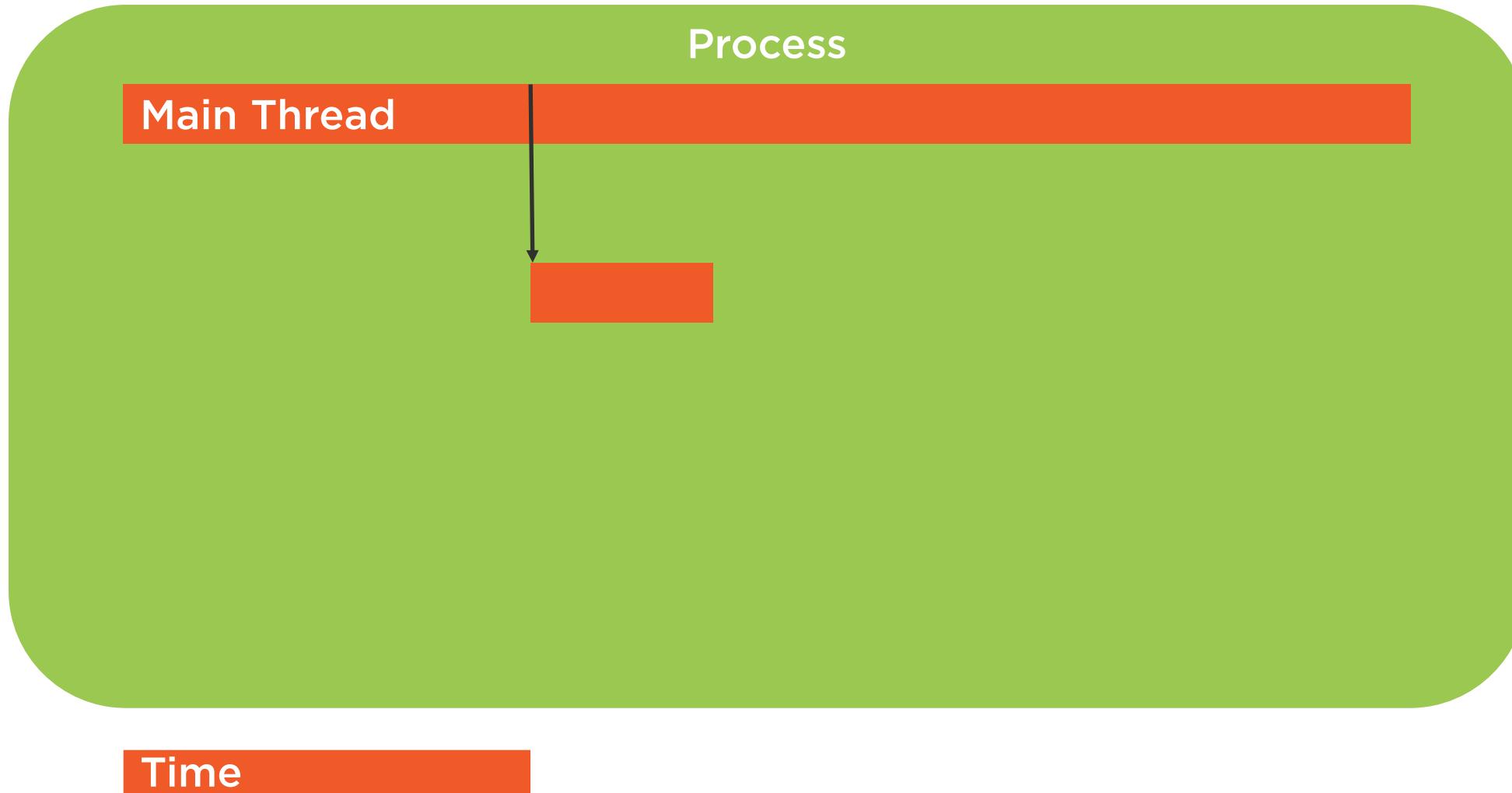
# Note Database Operations in Our App



# Database Interaction and Activity UI



# Database Interaction and Activity UI



# Database Interaction and Activity UI

## **Always avoid database action on UI thread**

- Don't request database connection
- Don't execute database operation

## **Can use a variety of threading solutions**

- Commonly use the AsyncTask class



# AsyncTask

## Implementing database interaction

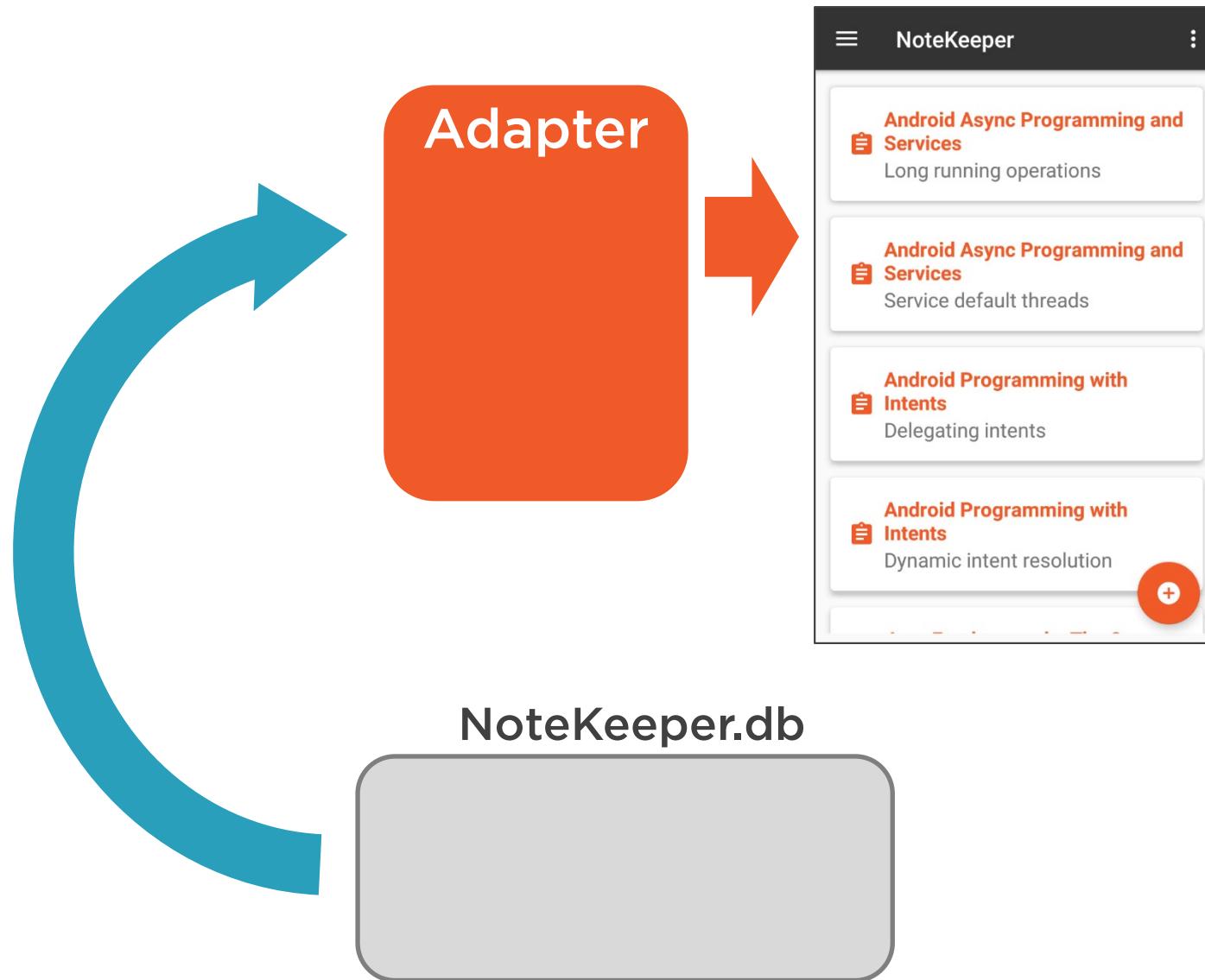
- Extend AsyncTask
- Override doInBackground method
- Add database code to doInBackground

## Performing interaction in background

- Call execute method
- AsyncTask will execute doInBackground on non-UI thread



# Note in Our Main Activity



# Our Database

note\_info

_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Delegating intents	PendingIntents are powerful...	android_intents



# Our Database

note\_info

_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Delegating intents	PendingIntents are powerful...	android_intents

course\_info

_id	course_id	course_title
1	android_intents	Android Programming with Intents
2	android_async	Android Async Programming and Services
3	java_lang	Java Fundamentals: The Java Language
4	java_core	Java Fundamentals: The Core Platform



# Joining Tables

query (

ON



```
Dynamic intent resolution | android_intents
```



# Column Names in Joins

**Column names may be ambiguous in a join**

- Tables may duplicate column names
- Need a way to disambiguate names

**Column names can be table qualified**

- Precede column name with table name
- Separate with a dot (i.e. period)

`course_id`

`course_id`



# Column Names in Joins

## When to use table qualified names

- Specifying columns in join statement
- Specifying desired list of columns

## When not to use table qualified names

- Don't pass to getColumnIndex



# Our note\_info Table

```
CREATE TABLE note_info  
    _id INTEGER ,  
    note_title TEXT ,  
    note_text TEXT ,  
    course_id TEXT )
```



# Our note\_info Table

Primary  
Key

	note_info			
Key	_id	note_title	note_text	course_id
	1	Dynamic intent resolution	Wow, intents allow components...	android_intents
	2	Anonymous classes	Anonymous classes simplify...	java_lang
	3	Delegating intents	PendingIntents are powerful...	android_intents
	4	Parameters	Leverage variable-length param...	java_lang
→	5	Long running operations	Foreground services can be tie...	android_async
	6	Serialization	Remember to include SerialVer...	java_core
	7	Service default threads	Did you know that by default A...	android_async
	8	Compiler options	The -jar option isn't compatible...	java_core



# Our note\_info Table

Index on  
note\_title

note_info				
	_id	note_title	note_text	course_id
	1	Dynamic intent resolution	Wow, intents allow components...	android_intents
	2	Anonymous classes	Anonymous classes simplify...	java_lang
	3	Delegating intents	PendingIntents are powerful...	android_intents
→	4	Parameters	Leverage variable-length param...	java_lang
	5	Long running operations	Foreground services can be tie...	android_async
	6	Serialization	Remember to include SerialVer...	java_core
	7	Service default threads	Did you know that by default A...	android_async
	8	Compiler options	The -jar option isn't compatible...	java_core



# Indexes

## **Helpful in a number of scenarios**

- Column appears in selection criteria
- Column is part of query ordering
- Column is part of a join

## **Indexes don't affect way query is written**

- Database automatically chooses whether to utilize index(es)



## Indexes

Indexes are sometimes created implicitly

- Adding a unique column constraint

```
CREATE TABLE course_info (
    _id INTEGER PRIMARY KEY,
    course_id TEXT UNIQUE NOT NULL,
    course_title TEXT NOT NULL )
```



# Indexes

## Indexes can be explicitly created

- Table can have as many as needed
- By default an index allows duplicates
- Can specify a uniqueness requirement
- Can span multiple columns

## Creating an index

- Use CREATE INDEX statement
- Provide name for index
- Identify table and one or more columns



# Joining Tables

query (

ON



note\_title

course\_title

Dynamic intent resolution

Android Programming with Intents

Anonymous classes

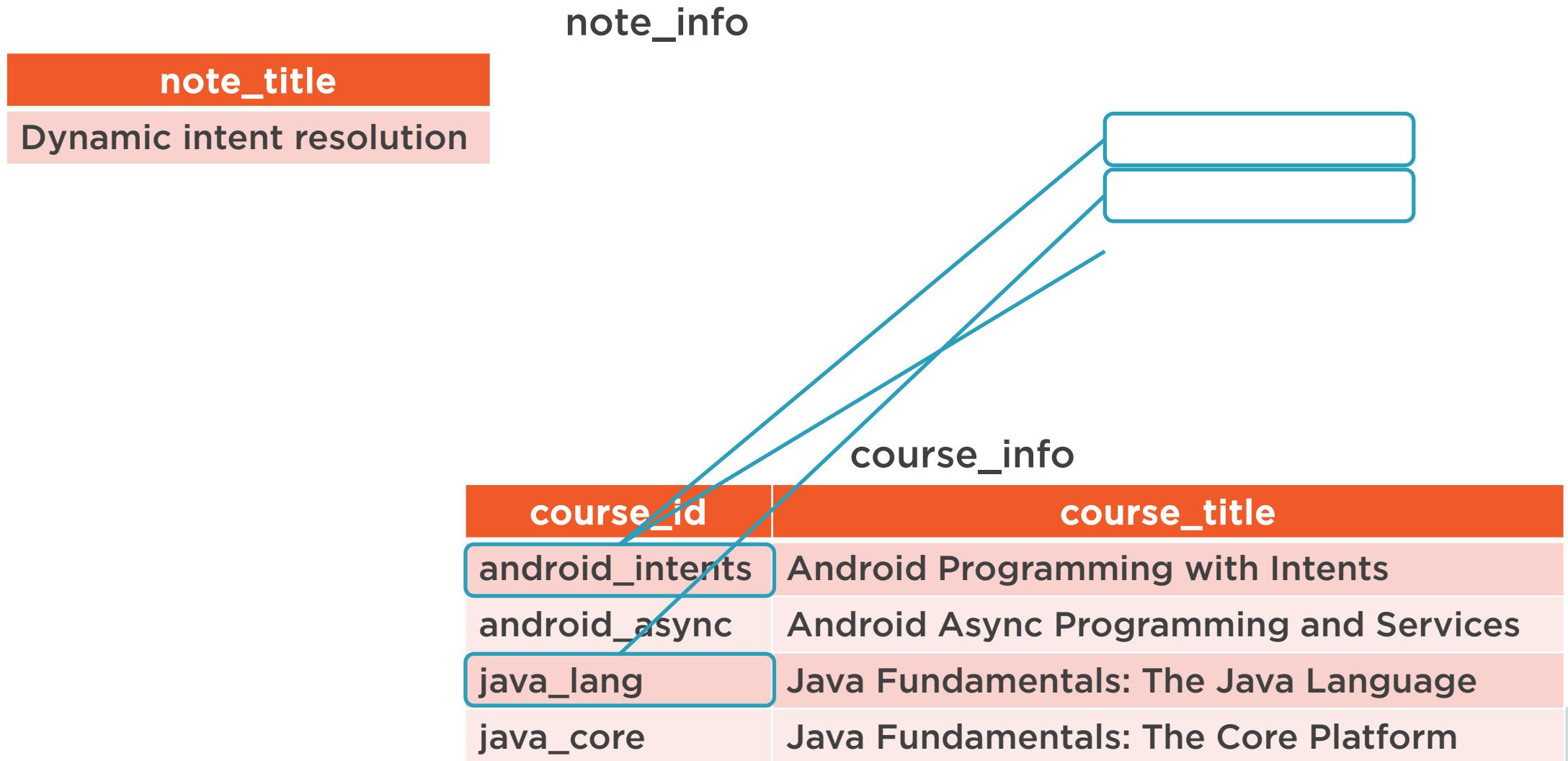
Java Fundamentals: The Java Language

Delegating intents

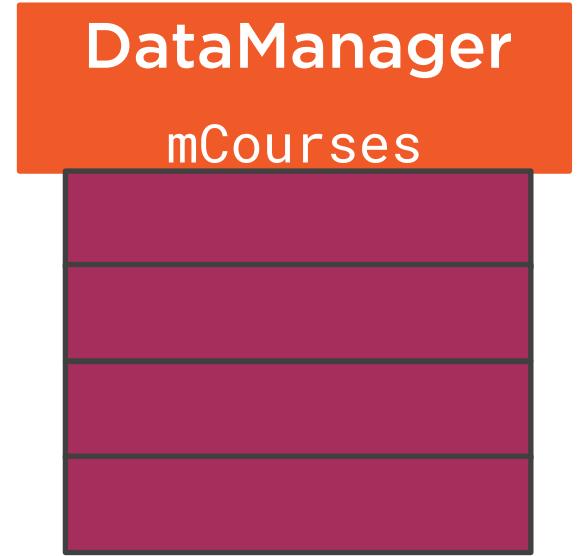
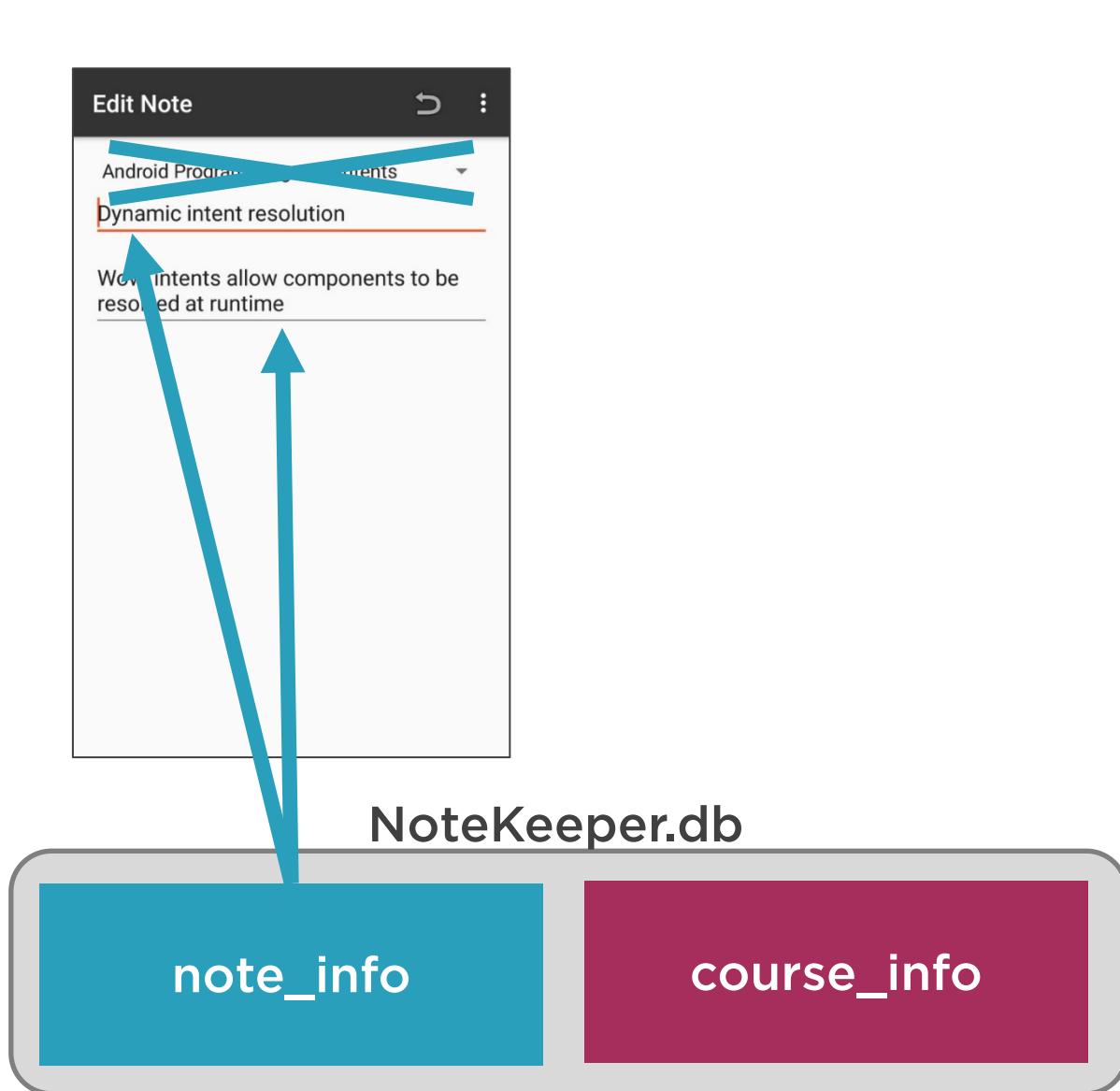
Android Programming with Intents



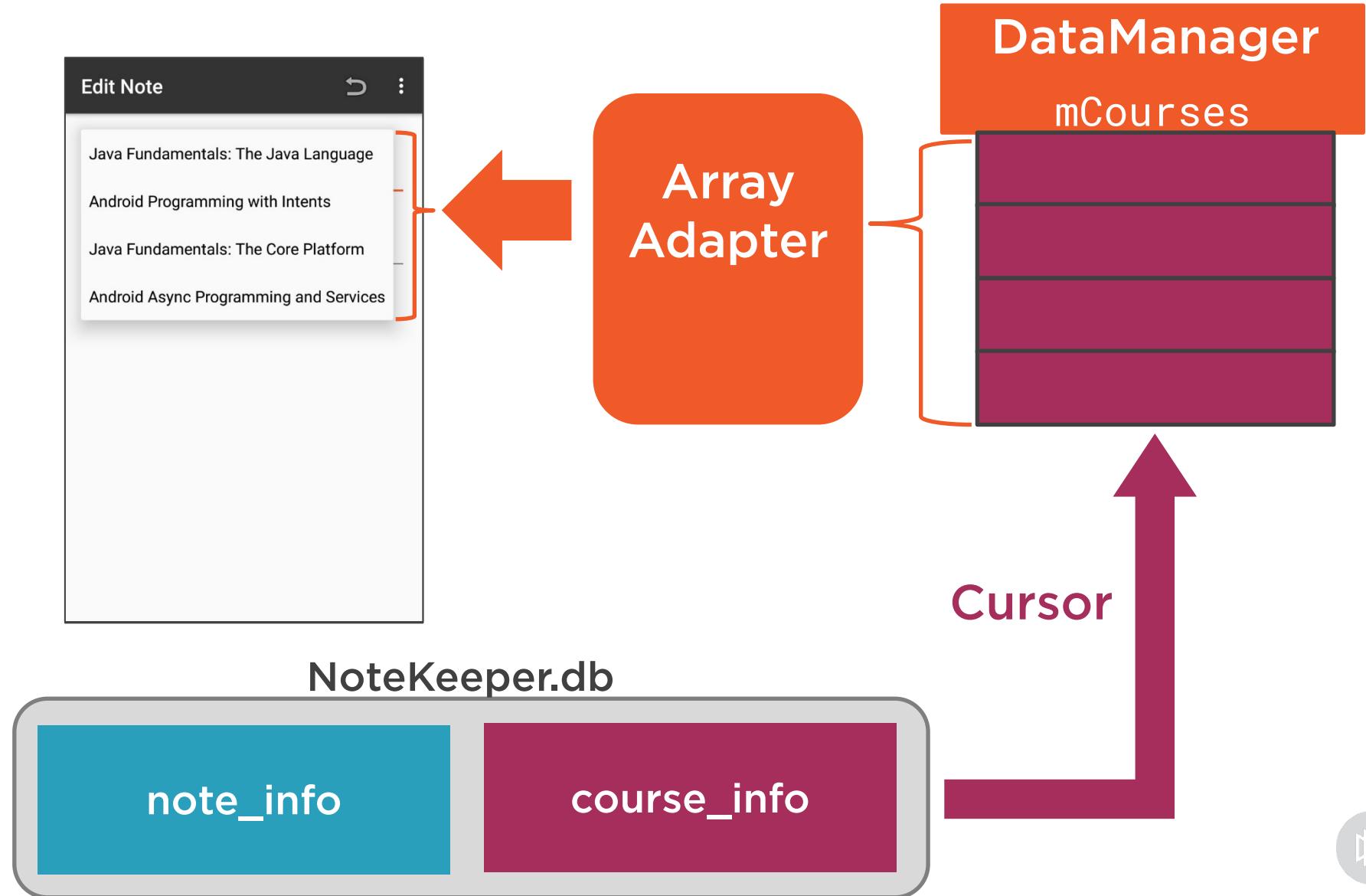
# Our Database



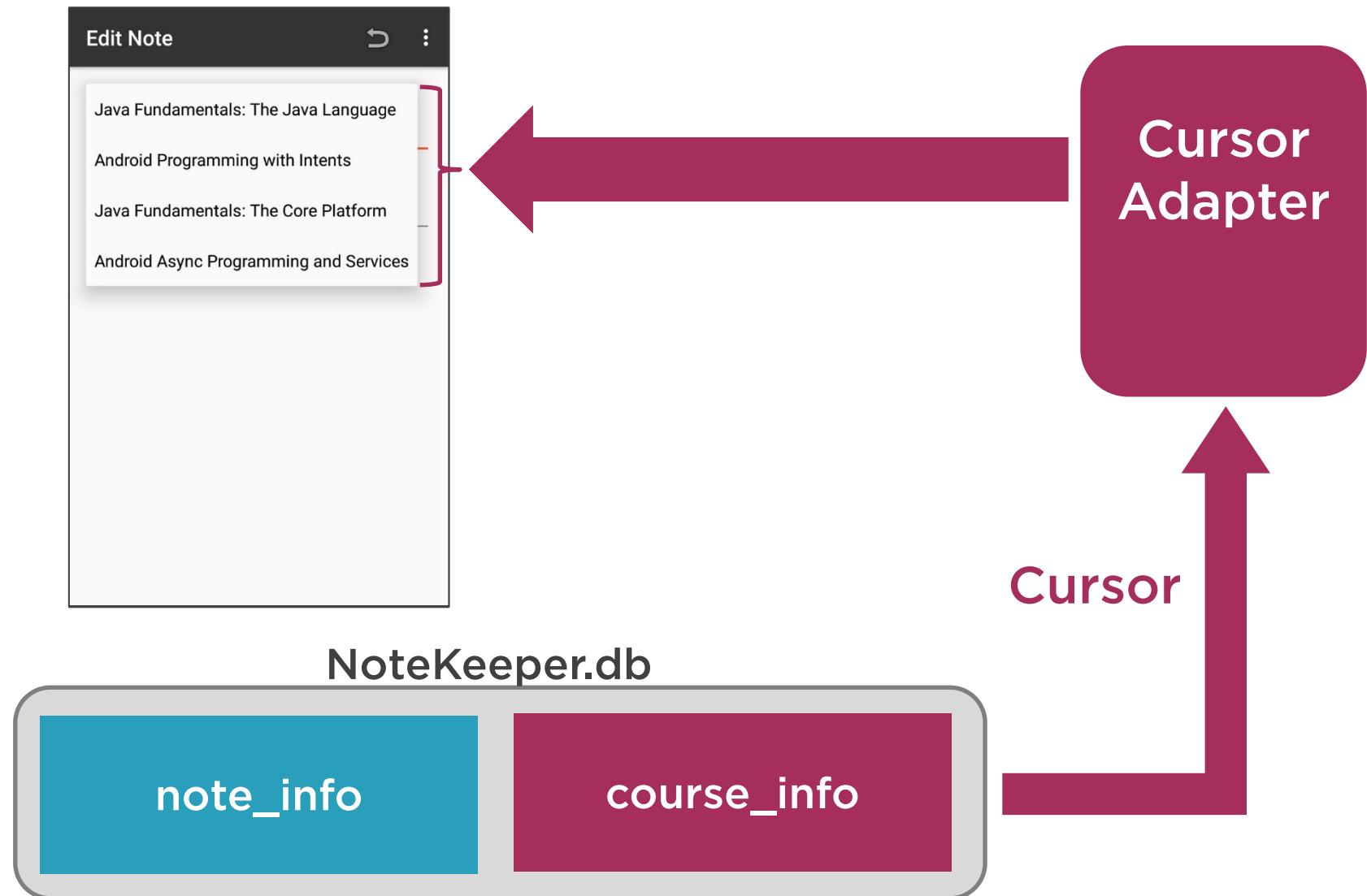
# Populating NoteActivity



# Populating NoteActivity



# Populating NoteActivity



# CursorAdaptor

## **Populates Spinner/ListView from Cursor**

- Manages Cursor interaction
- Provides methods for creating views
- Provides methods for binding views



# CursorAdaptor

## Associating Cursor

- Can use constructor
- Can use changeCursor method

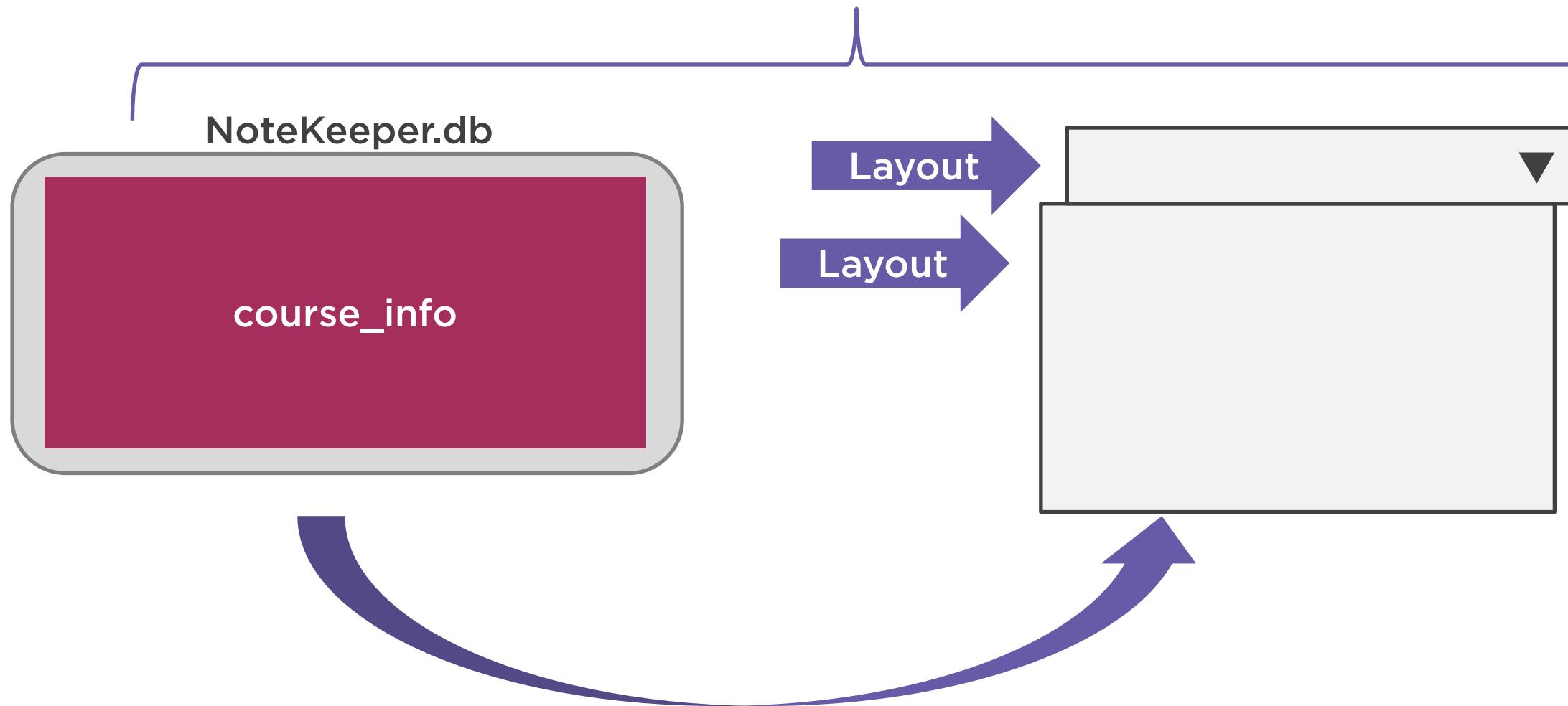
## Using changeCursor method

- Associates new cursor
- Closes previous cursor
- Call with null when done with adaptor



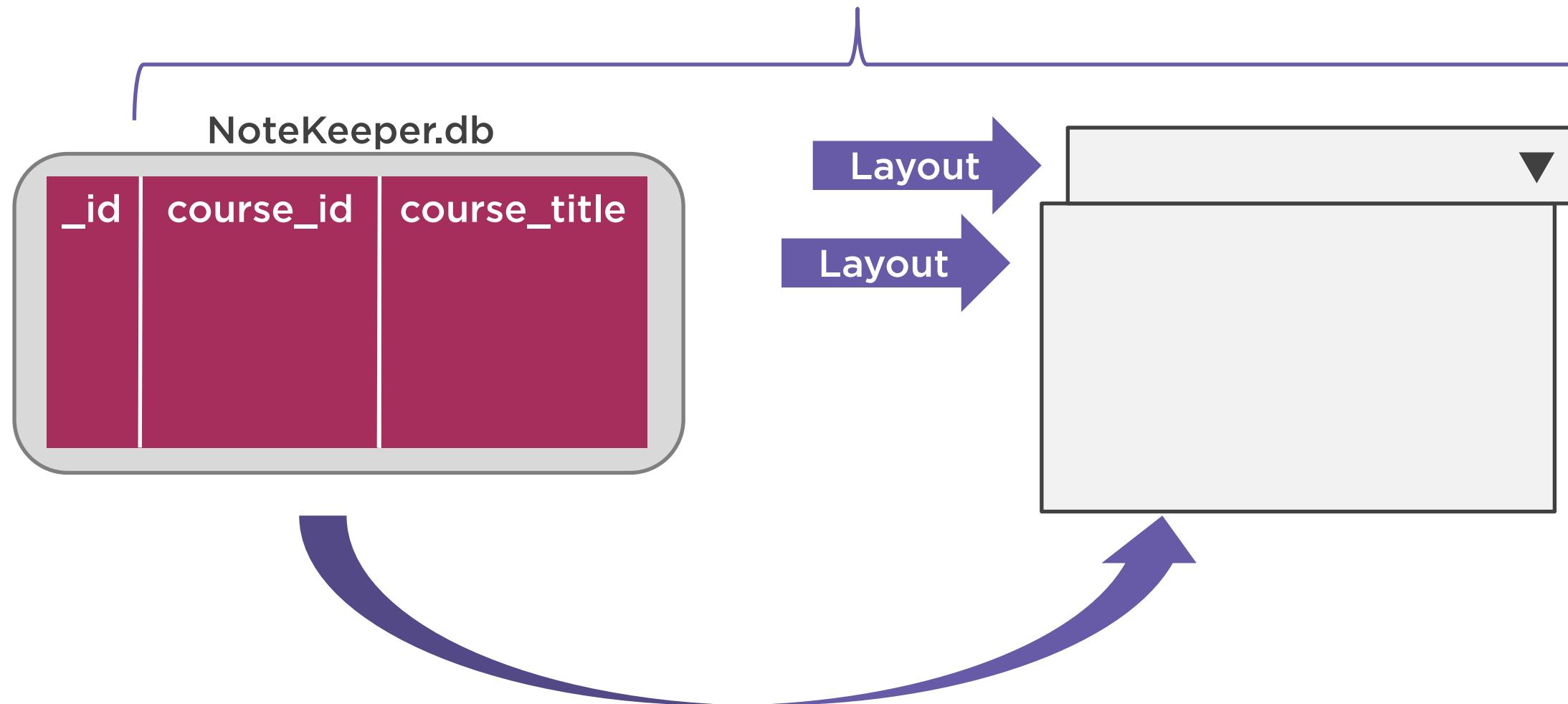
# Populating a Spinner

## CursorAdapter



# Populating a Spinner

## CursorAdapter



# CursorAdaptor

**CursorAdaptor is extremely flexible**

- You have to handle display details
- You must add code to create views
- You must add code to populate views



# Simple CursorAdaptor

## **Extends CursorAdapter**

- Leverages same cursor management

## **Provides default display handling**

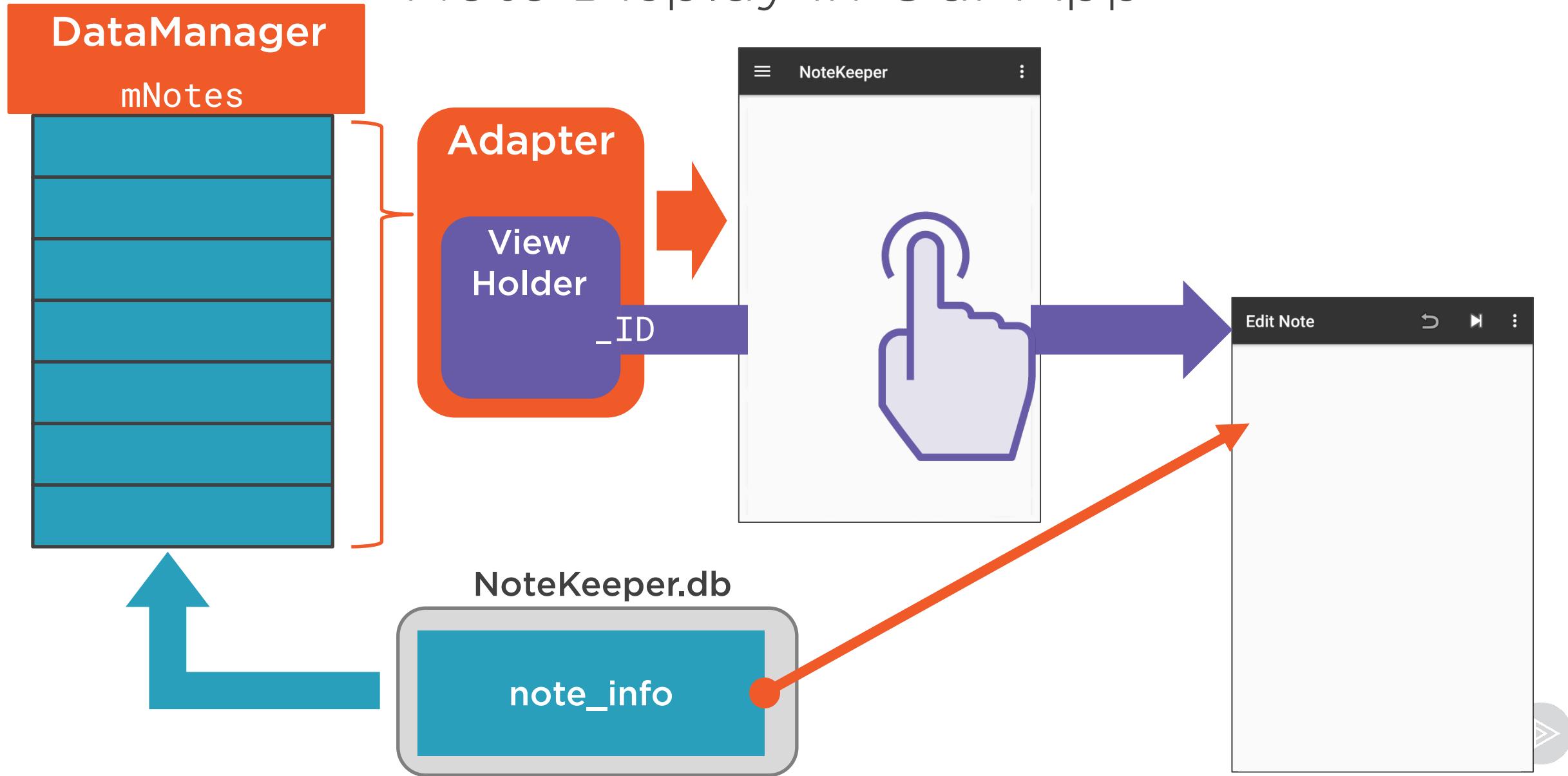
- Works for most common scenarios

## **You provide the basic display information**

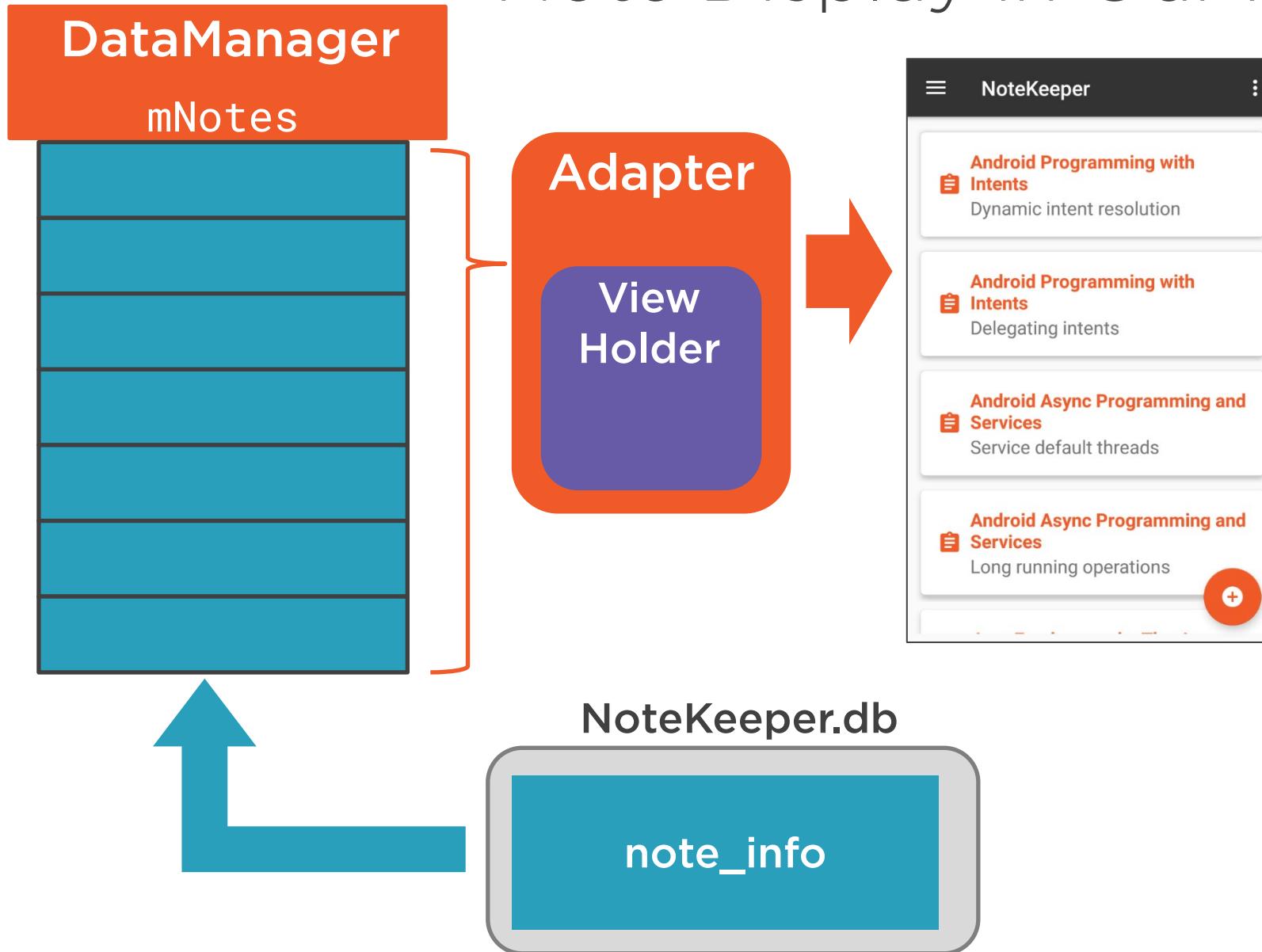
- Display resources
- Map of column names to view ID's



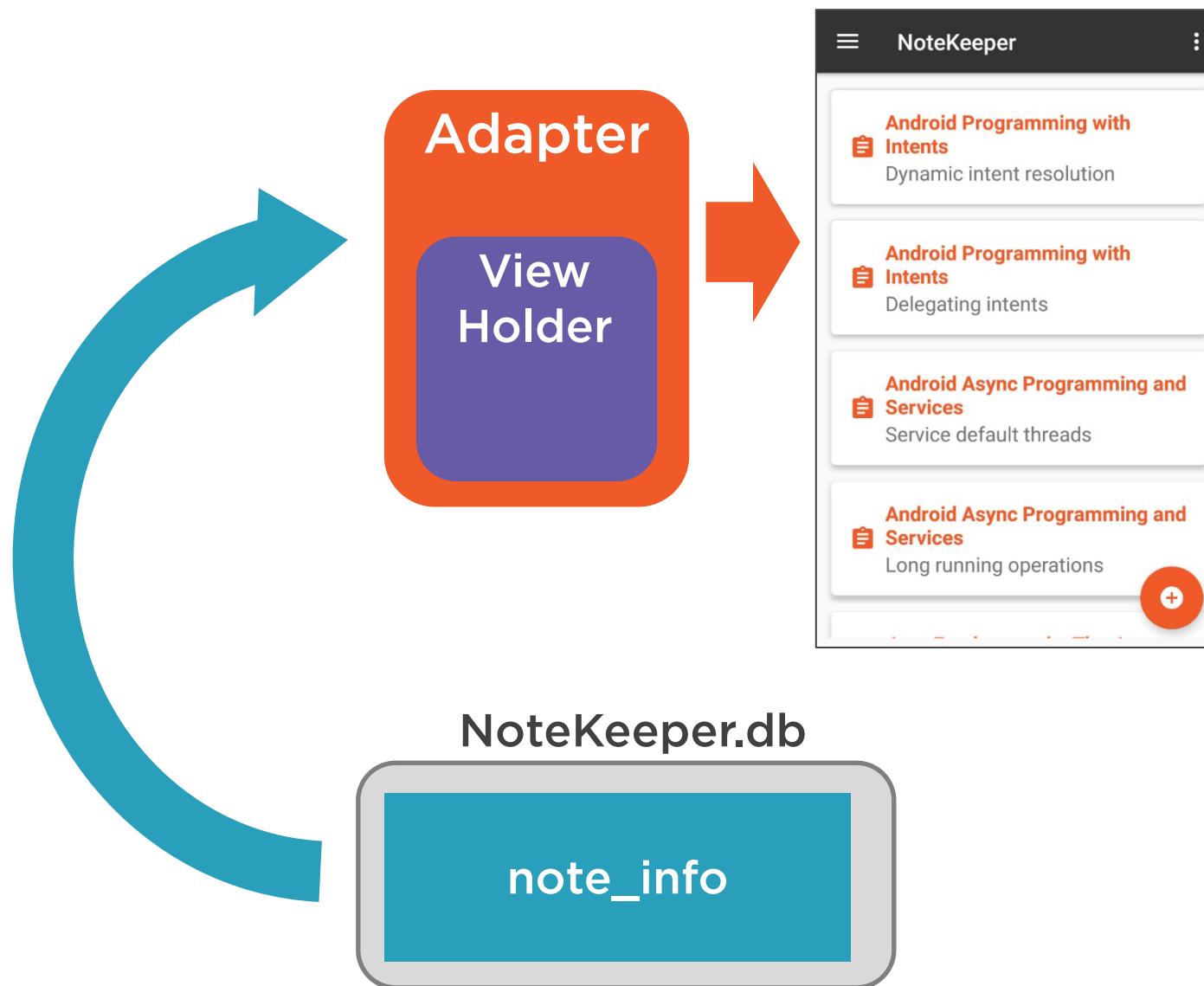
# Note Display in Our App



# Note Display in Our App



# Note Display in Our App



# RecyclerView Adapter With a Cursor

## Adapter data source

- Use cursor as data source
- Allow passing cursor to constructor
- Allow changing associated cursor



# RecyclerView Adapter With a Cursor

## Adapter view management

- Create ViewHolder instances
- RecyclerView manages as a pool

## Role of each ViewHolder instance

- Holds references to contained views
- Handles view interaction



# RecyclerView Adapter With a Cursor

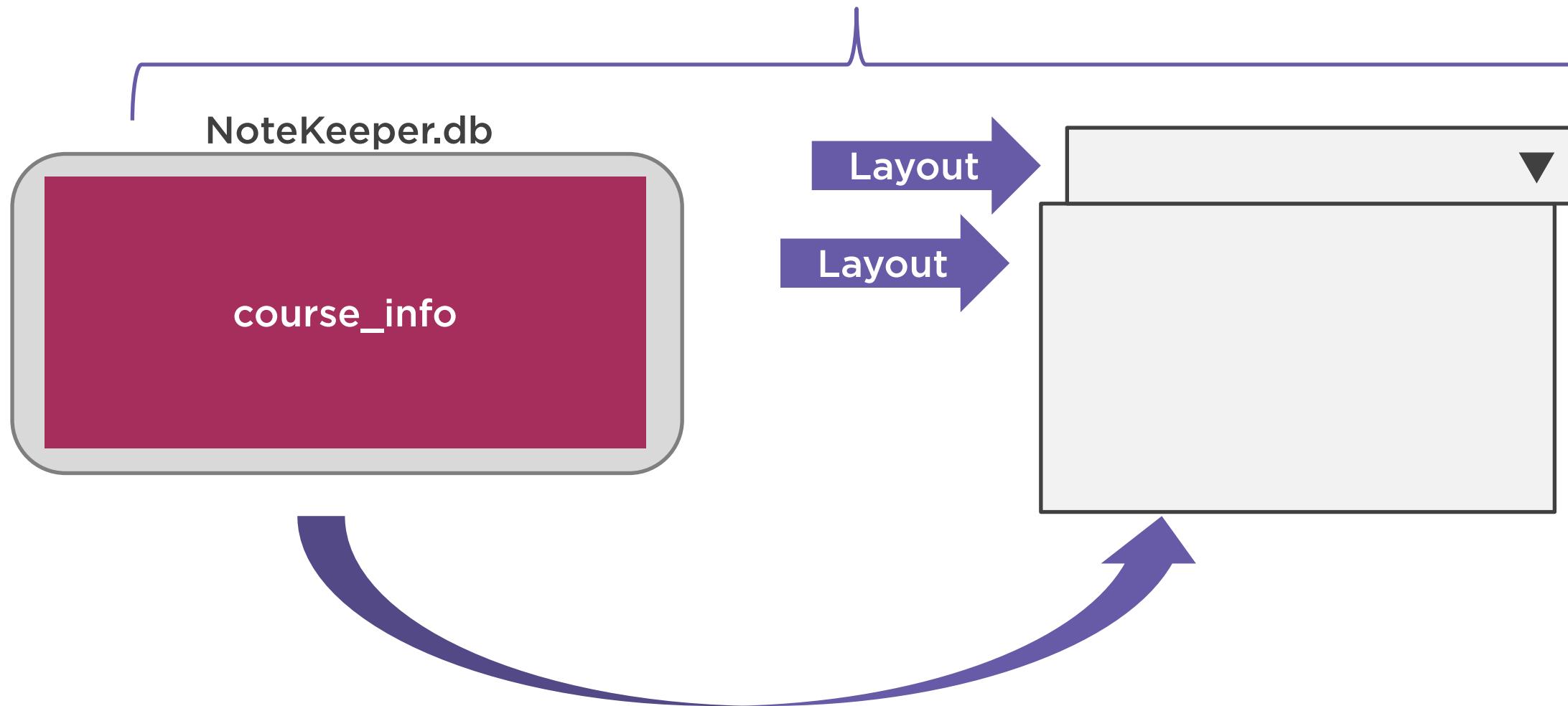
## Adapter data display

- Move cursor to a specific position
- Retrieve desired column values
- Set display values using ViewHolder



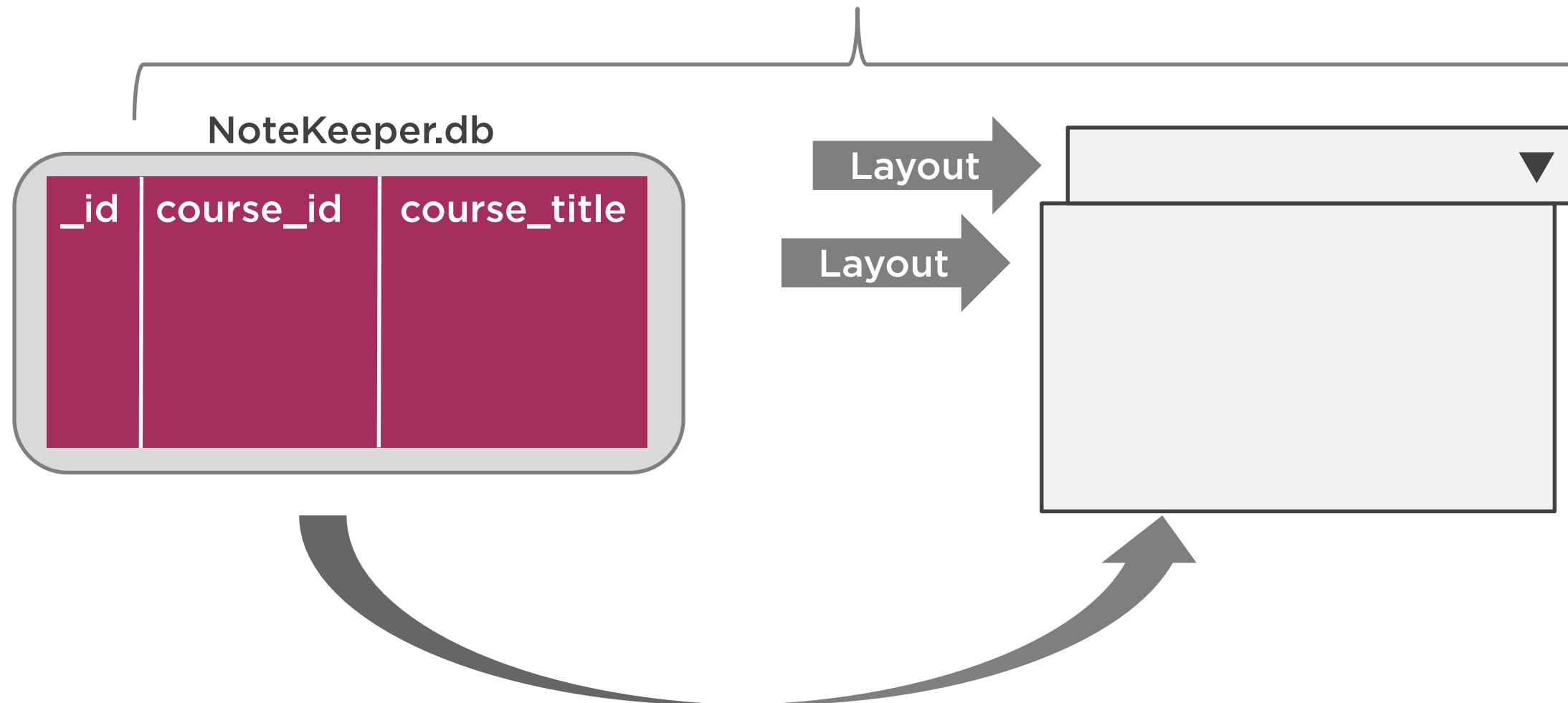
# Populating a Spinner

## CursorAdapter



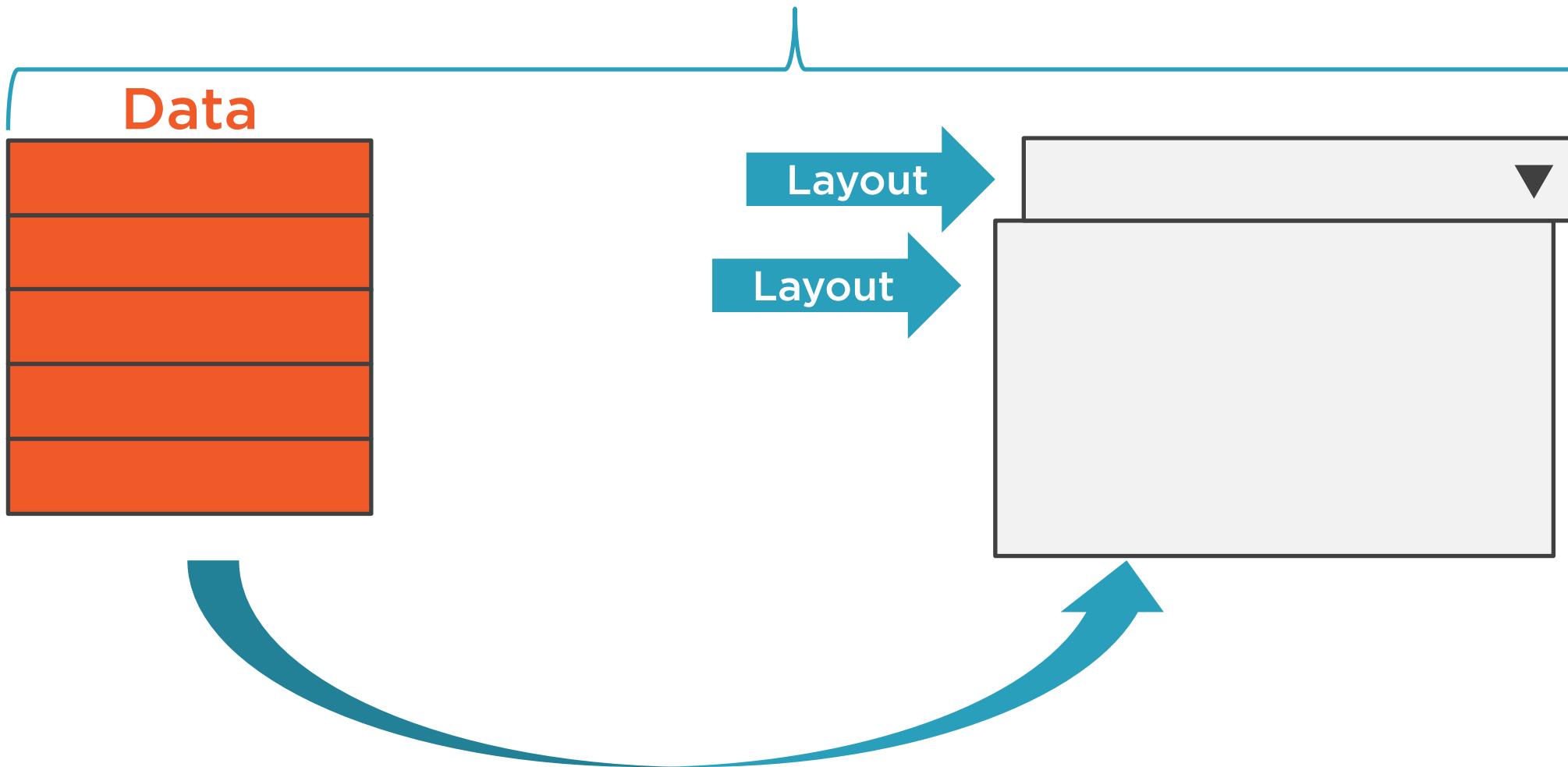
# Populating a Spinner

## CursorAdapter

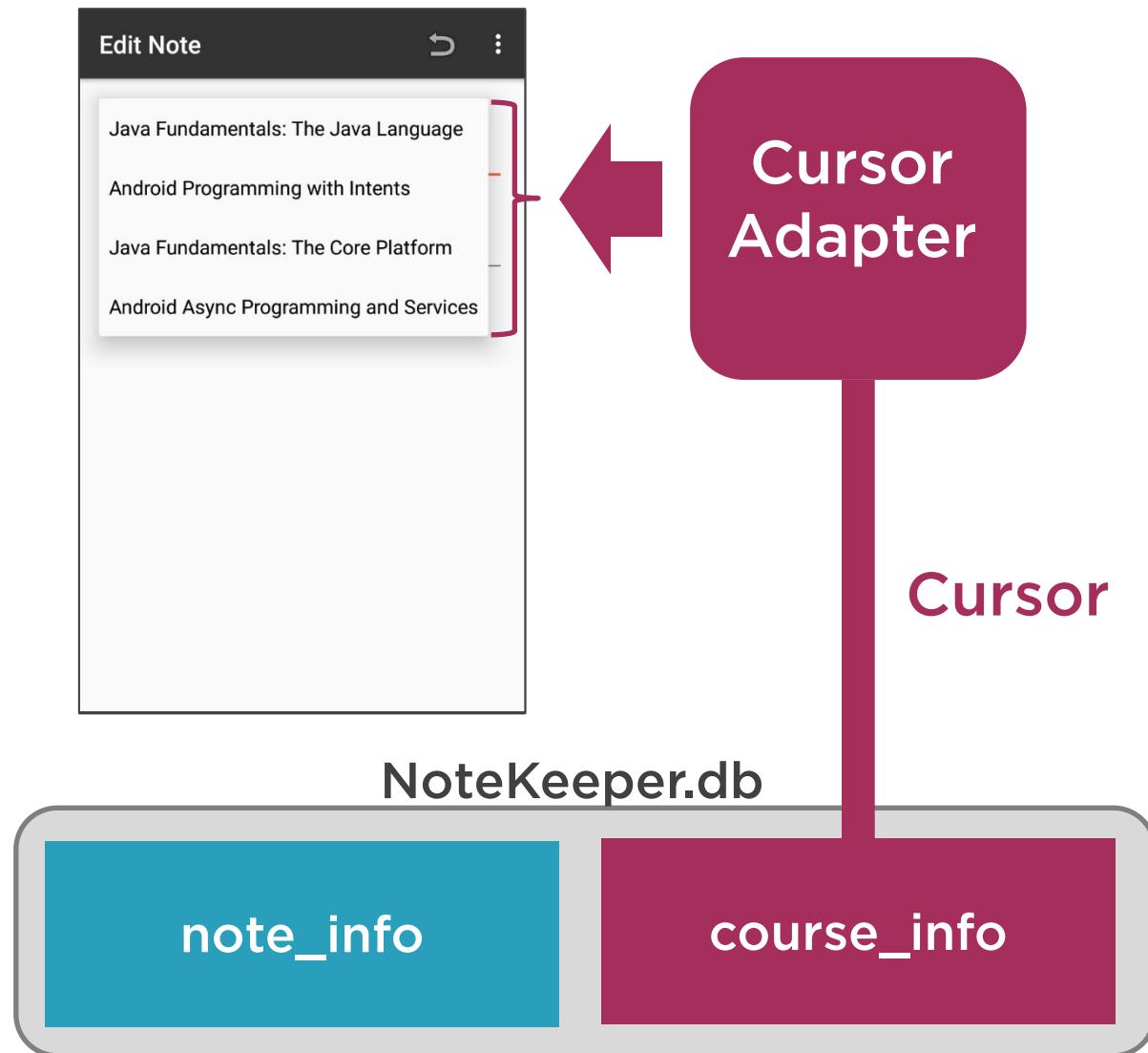


# Populating a Spinner

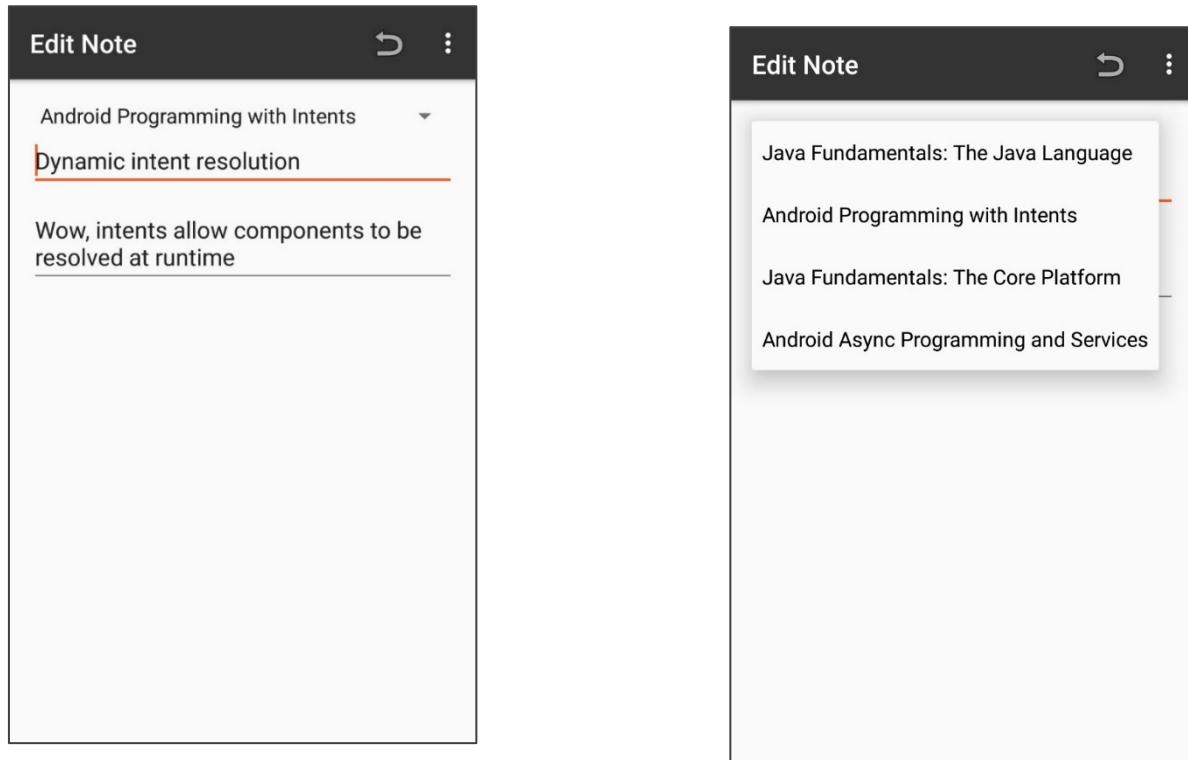
## Adapter



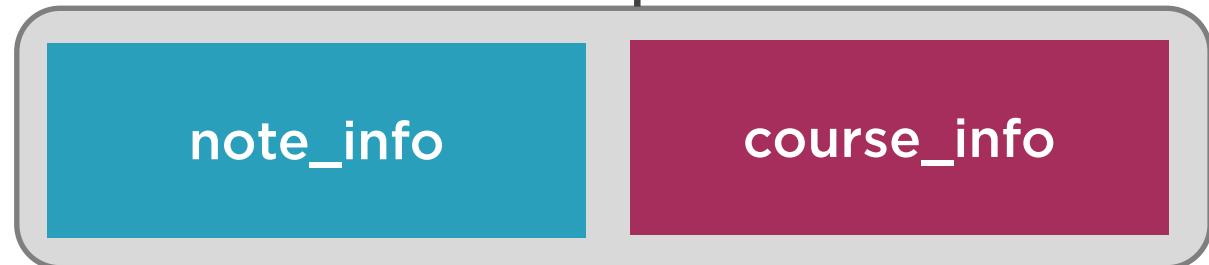
# Populating NoteActivity



# Note Display in Our App



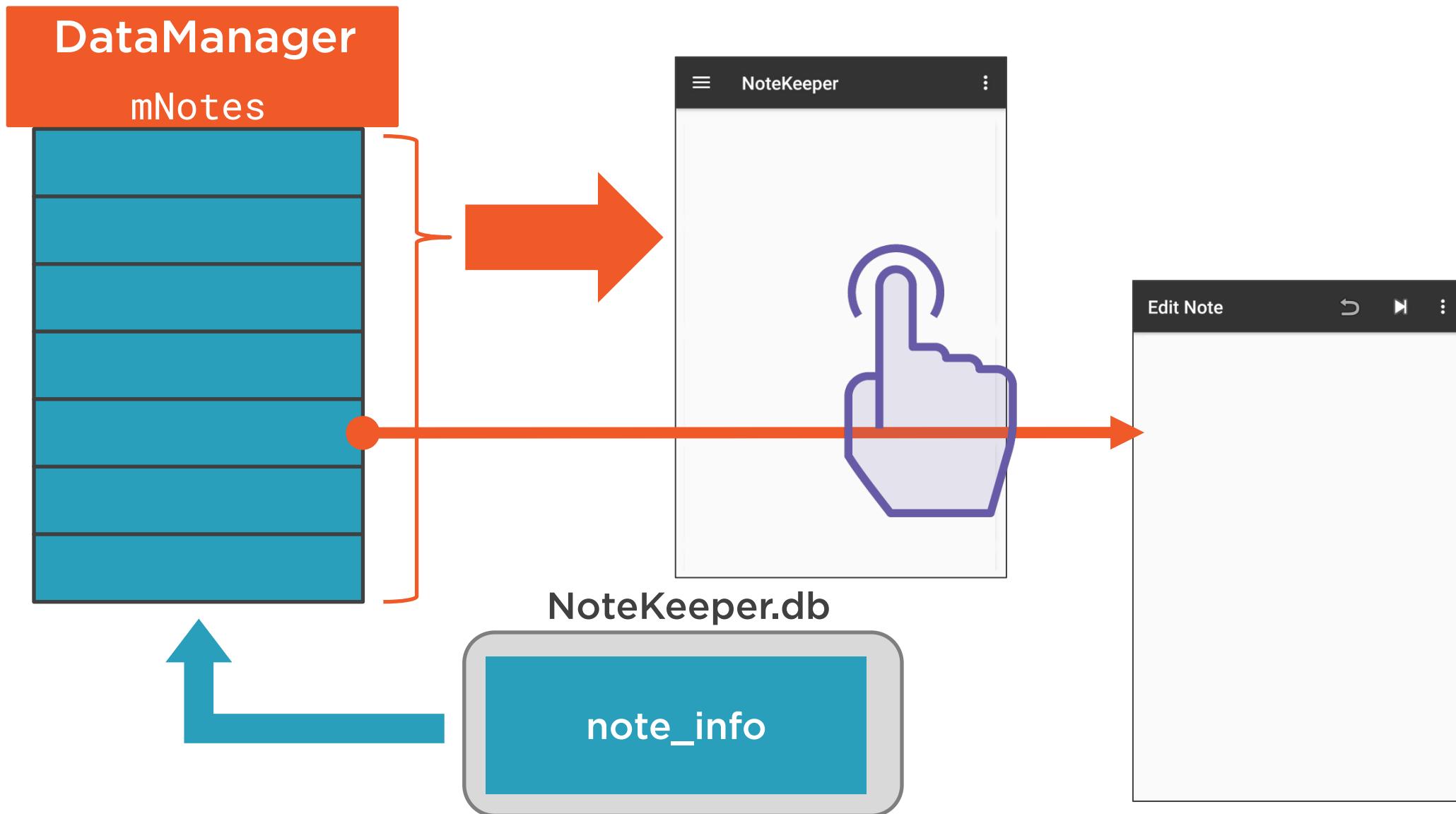
NoteKeeper.db



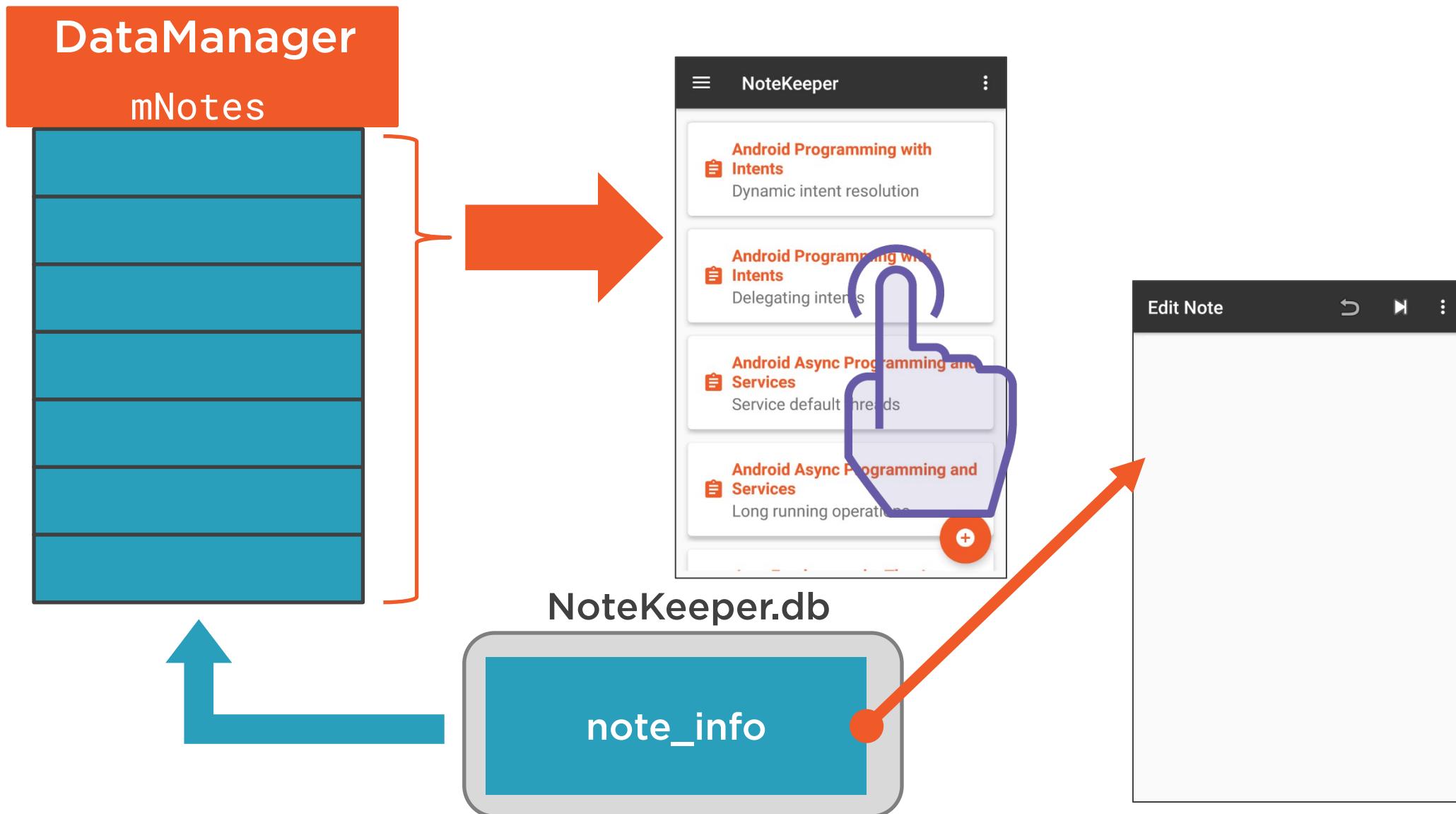
# Note Display in Our App



# Note Display in Our App



# Note Display in Our App



# Row Selection

**Often want only a subset of table rows**

- Let the database do the filtering work
- Can pass selection criteria to query



## Row Selection

### Parts of a simple selection

- Column name
- Operator
- Value

`course_id = "android_intents"`



# Row Selection

## Common operators

- = or ==
- != or <>
- >
- <
- >=
- <=
- LIKE



# Row Selection

## Common uses of LIKE

- note\_title LIKE “dynamic%”
  - Rows with a note\_title value that starts with dynamic
- note\_title LIKE “%intent%”
  - Rows with a note\_title value that contains intent



# Row Selection

## AND operator

- Combines two conditions
- True result only if both are true

```
course_id = "android_intents"  
          AND note_title LIKE "dynamic%"
```

Rows with both a course\_id value of android\_intents and a note\_title value that starts with dynamic



# Row Selection

## OR operator

- Combines two conditions
- True result when one or both are true

`course_id = "android_intents"`

`OR note_title LIKE "dynamic%"`

Rows with either a `course_id` value of `android_intents` or a `note_title` value that starts with `dynamic`



# Row Selection Parameters

## **Selection passed to query in two parts**

- Selection clause as a string
  - Uses ?'s as value position holders
- Selection value as a string array
  - Values replace ?'s in order



# Row Selection Parameters

```
Cursor queryCourse(String courseId) {  
    String selection =  
    String[] selectionArgs =  
    // return result of query with selection and selectionArgs  
}
```



# Row Selection Parameters

```
Cursor queryNote(String courseId, String titleStart) {  
    String selection =  
    String[] selectionArgs = { };  
    // return result of query with selection and selectionArgs  
}
```



# Row Selection Parameters

## Benefits of separating selection and values

- Protects against SQL attacks
- In some cases helps query performance



# Passing Row Information Between Activities

**Activities often share information**

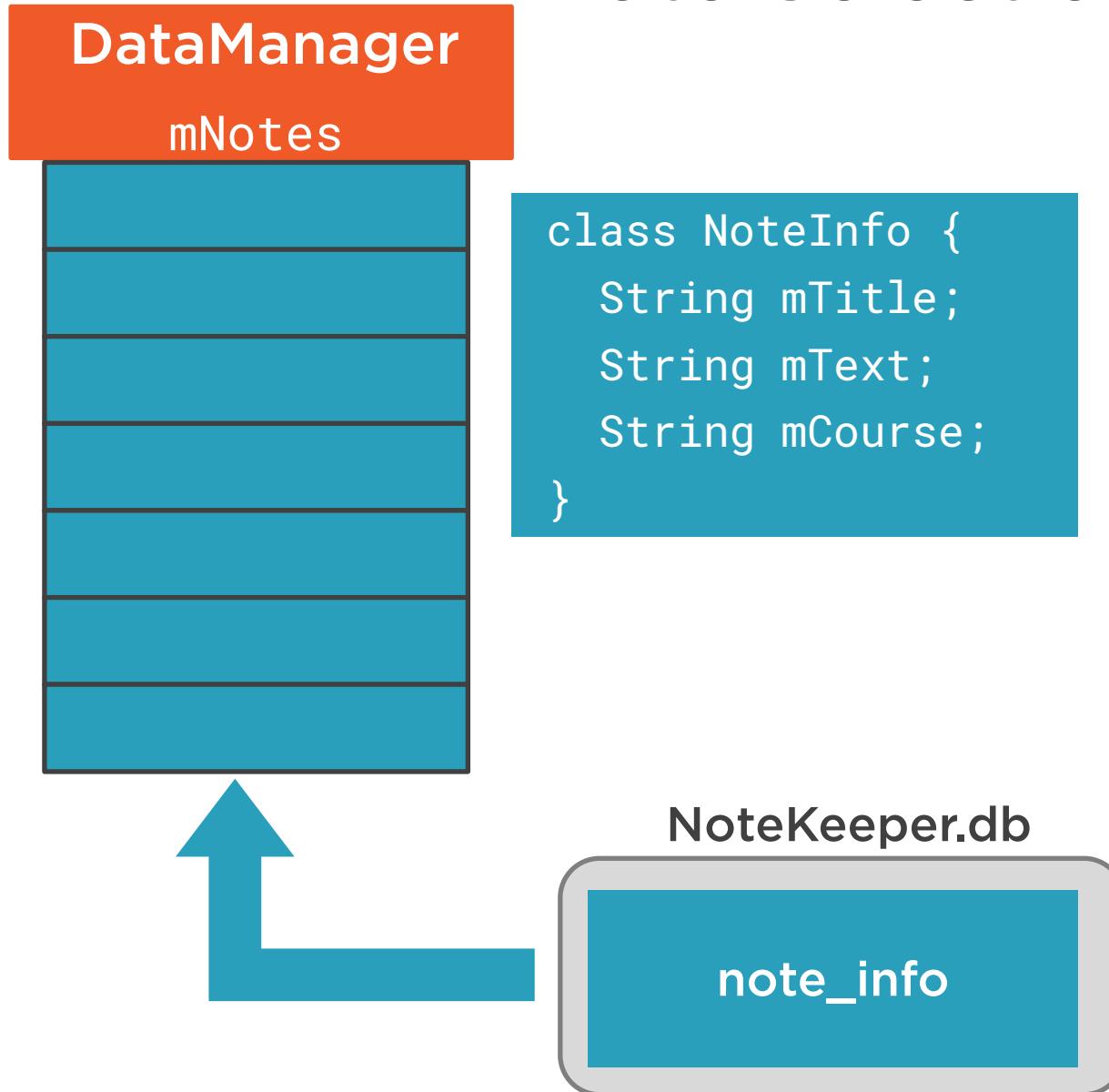
- Passed in intent extras

**Leverage unique row identifier**

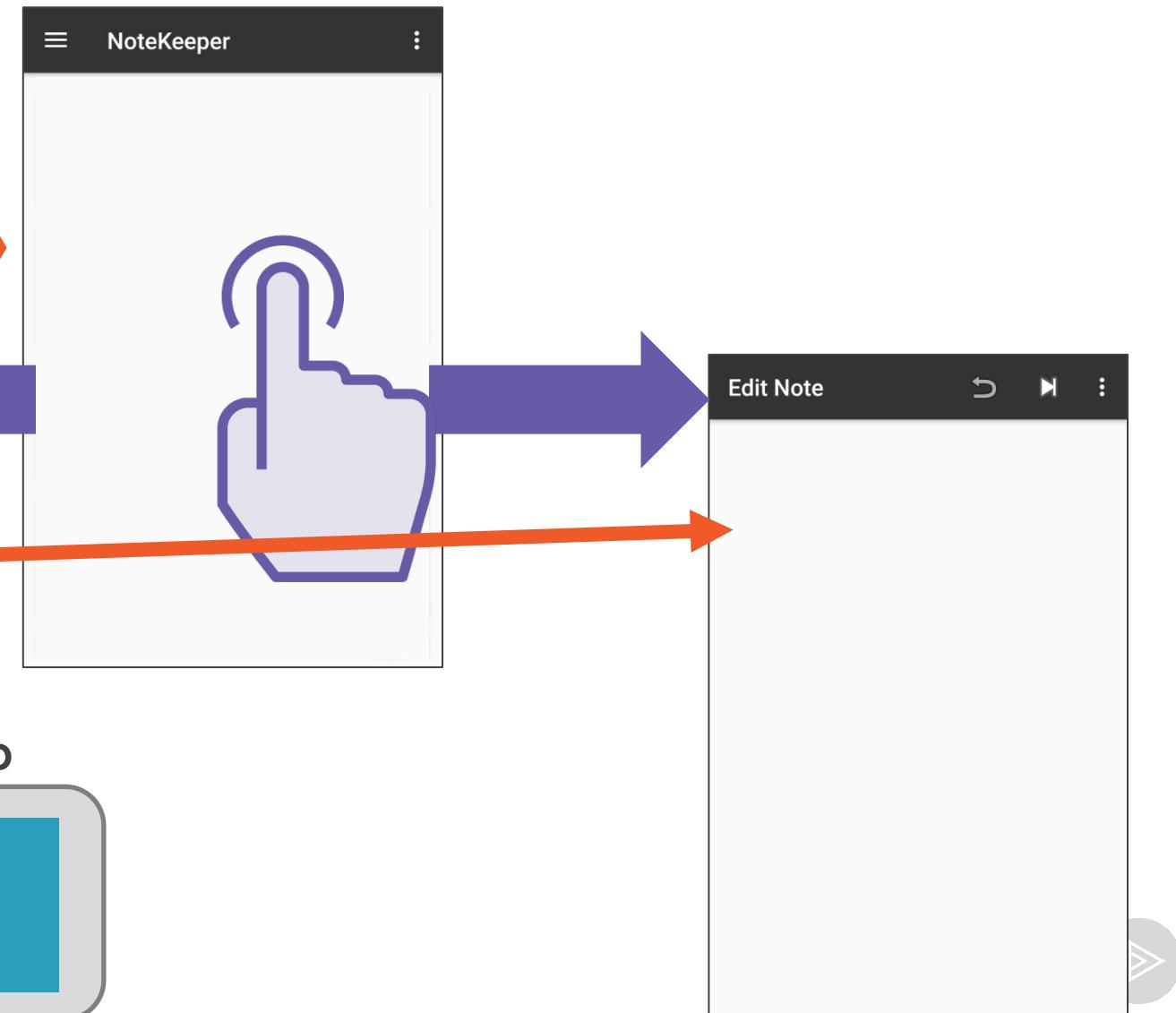
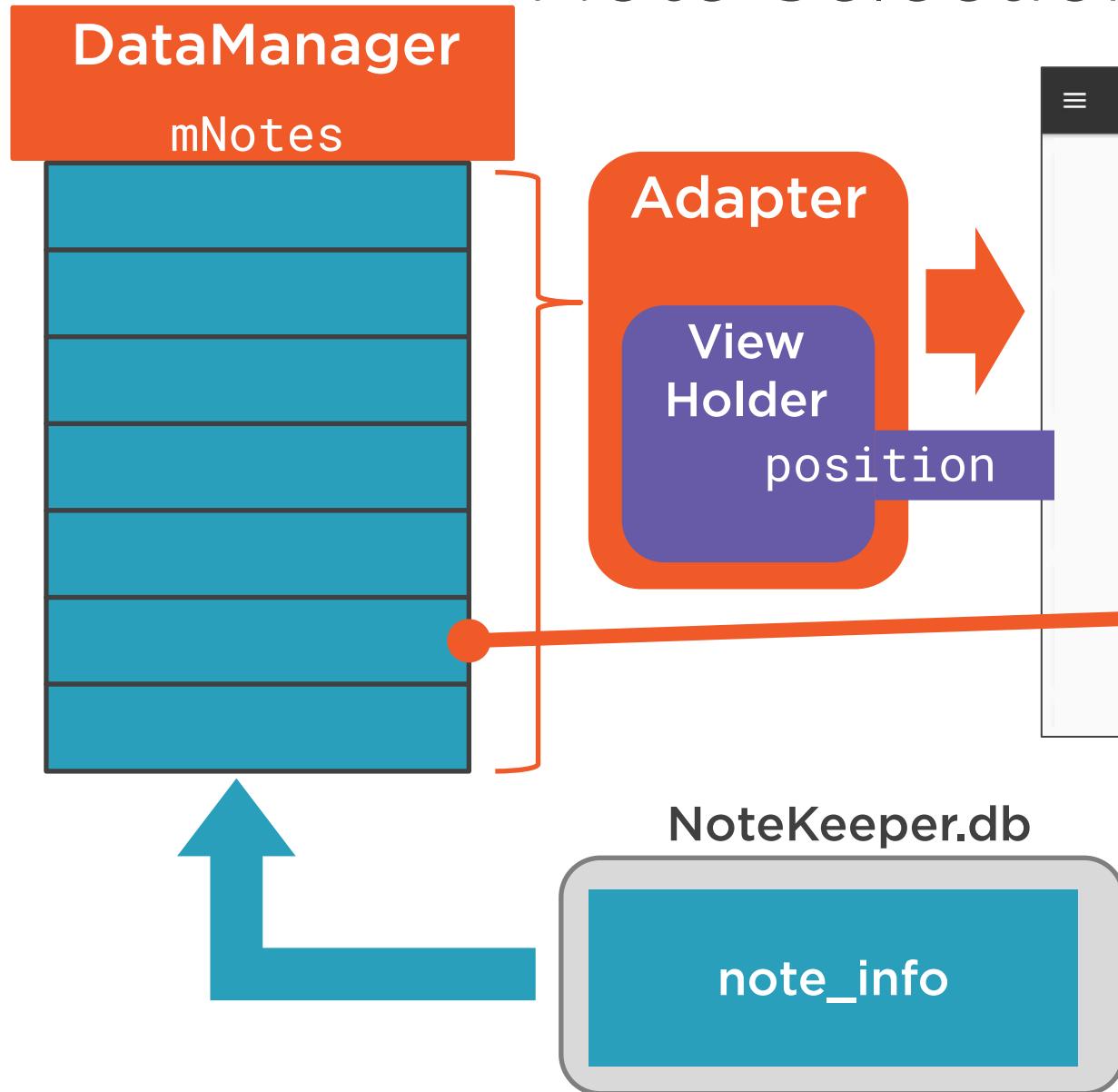
- BaseColumns.\_ID
- Most efficient way to find row
- Provides definitive row identity



# Note Selection in Our App



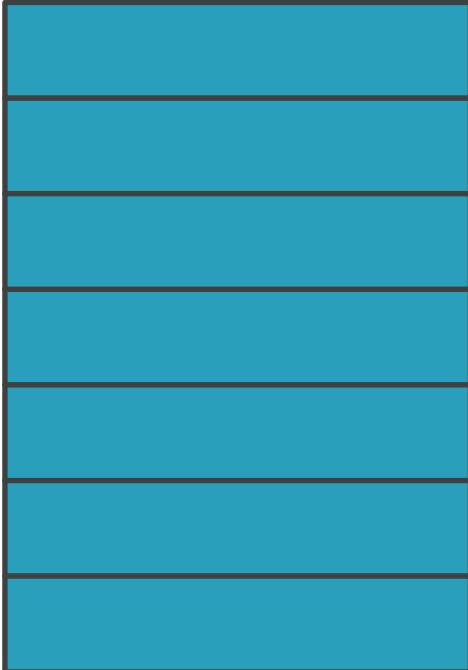
# Note Selection in Our App



# Note Selection in Our App

**DataManager**

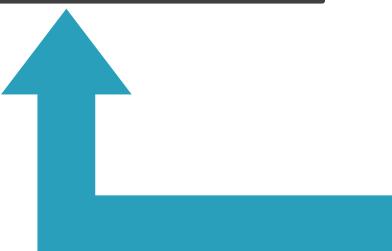
**mNotes**



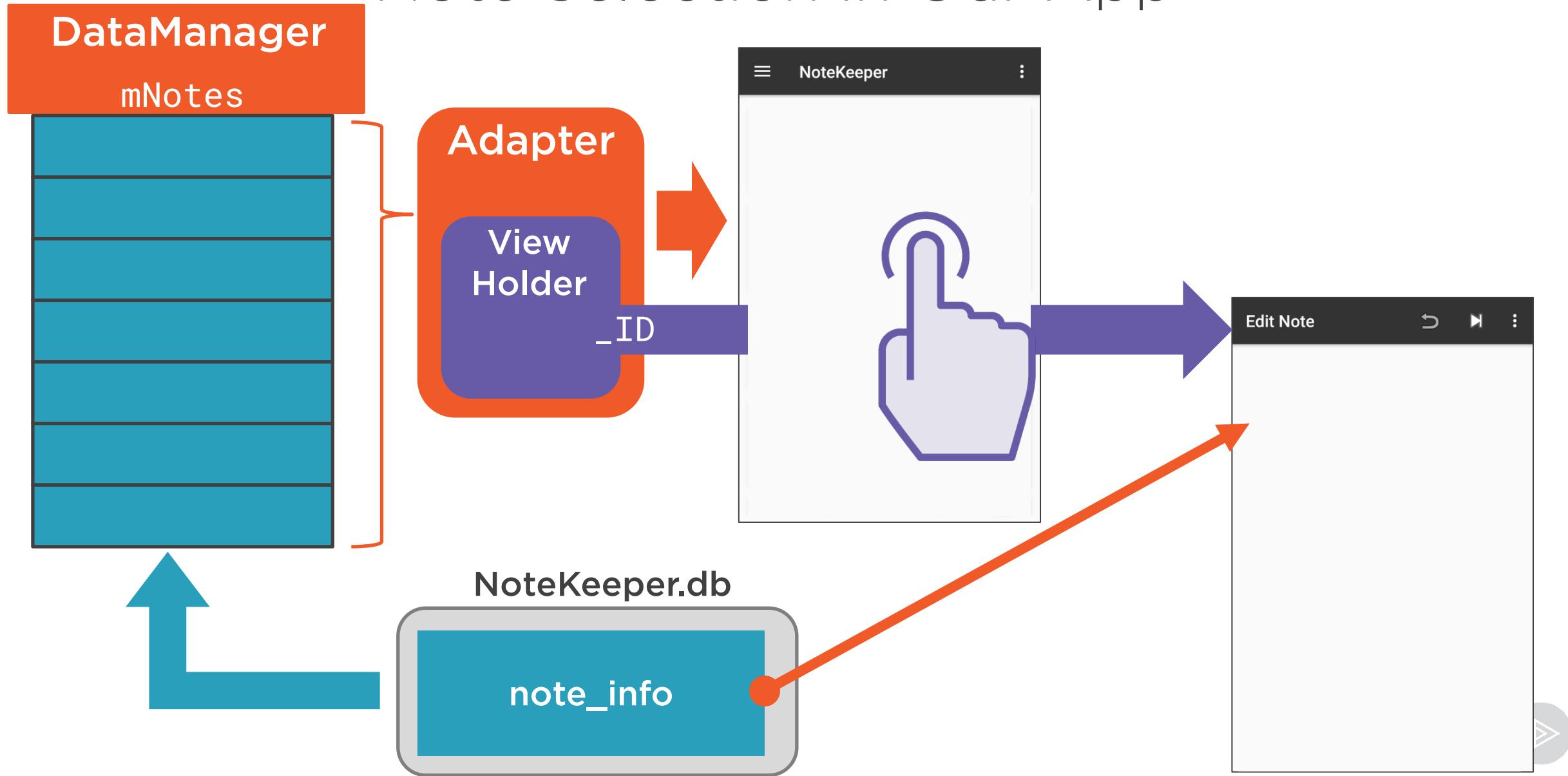
```
class NoteInfo {  
    String mTitle;  
    String mText;  
    String mCourse;  
    int mId;  
}
```

**NoteKeeper.db**

**note\_info**



# Note Selection in Our App



# Data\_for\_3

_id	city	age	user_name
1	Farmington	25	utah_girl
2	farmington	30	tech_time
3	orlando	52	ap_holder
4	Farmington	41	mtn_air
5	Los Angelos	38	sun_surfer

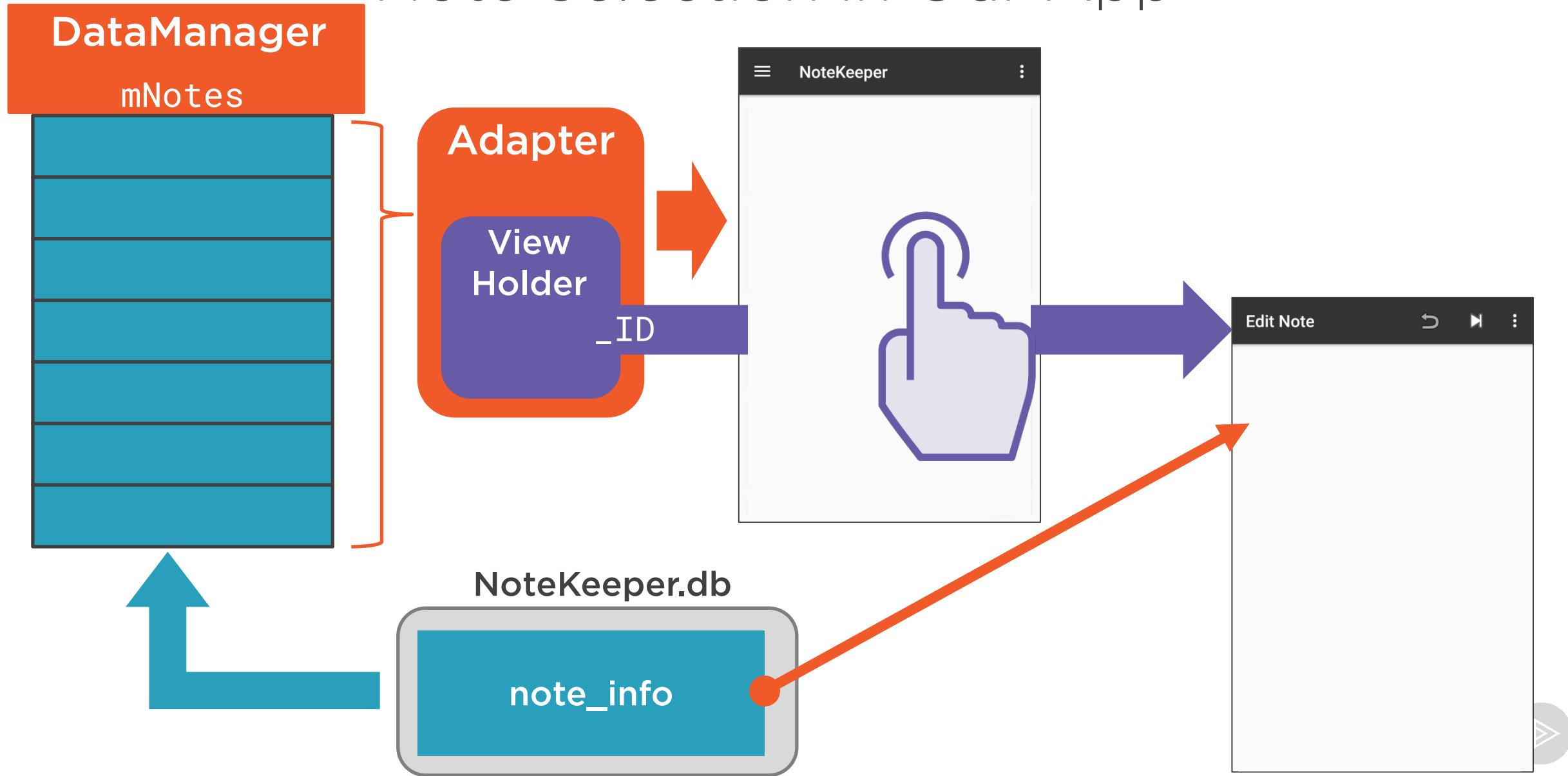


# Data\_for\_4

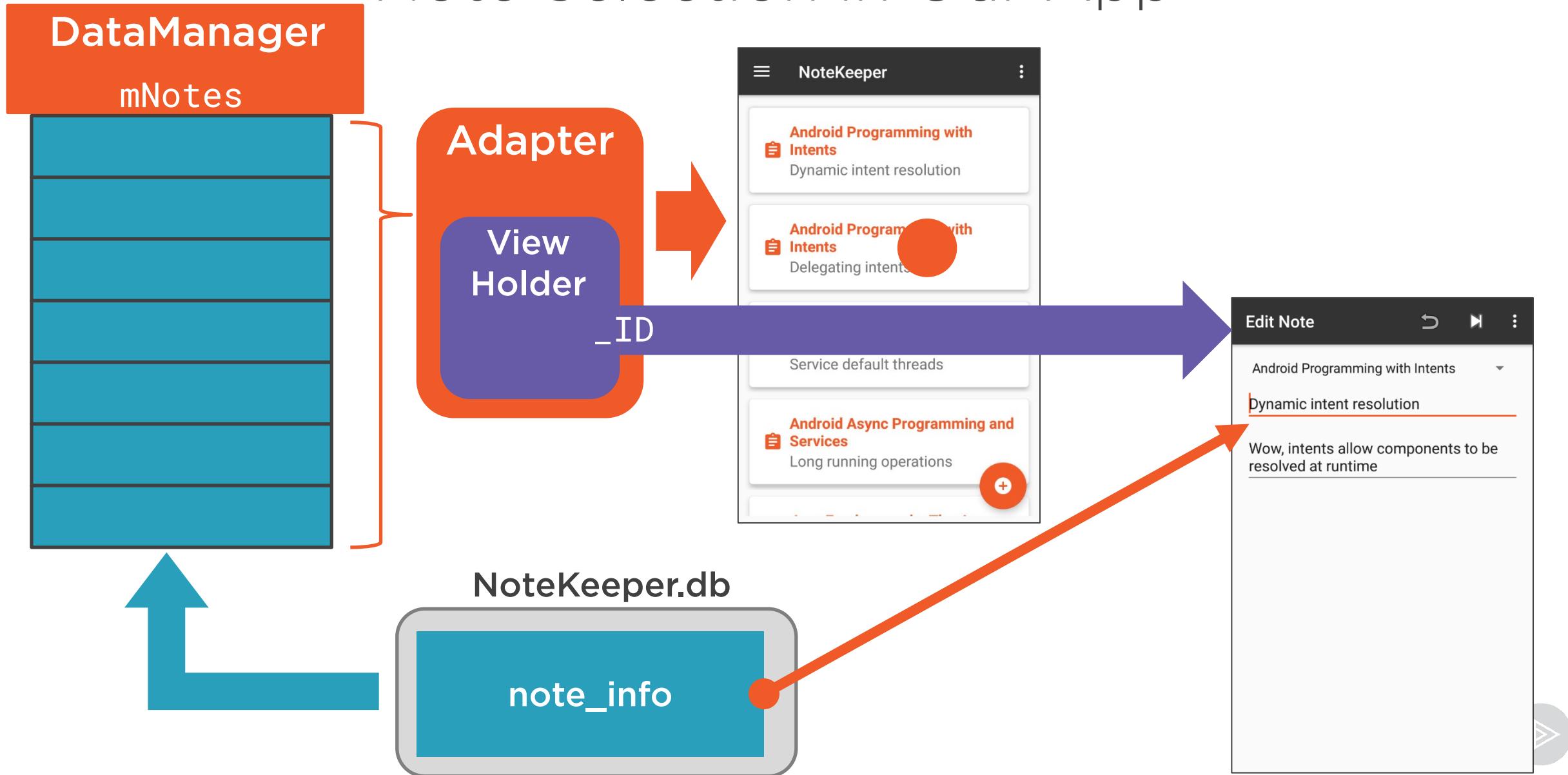
_id	city	age	user_name
1	New London	25	nh_girl
2	Newark	30	city_slicker
3	Boskanew	52	low_profile
4	North Newfieds	41	mtn_time
5	South New Durham	38	cool_breeze



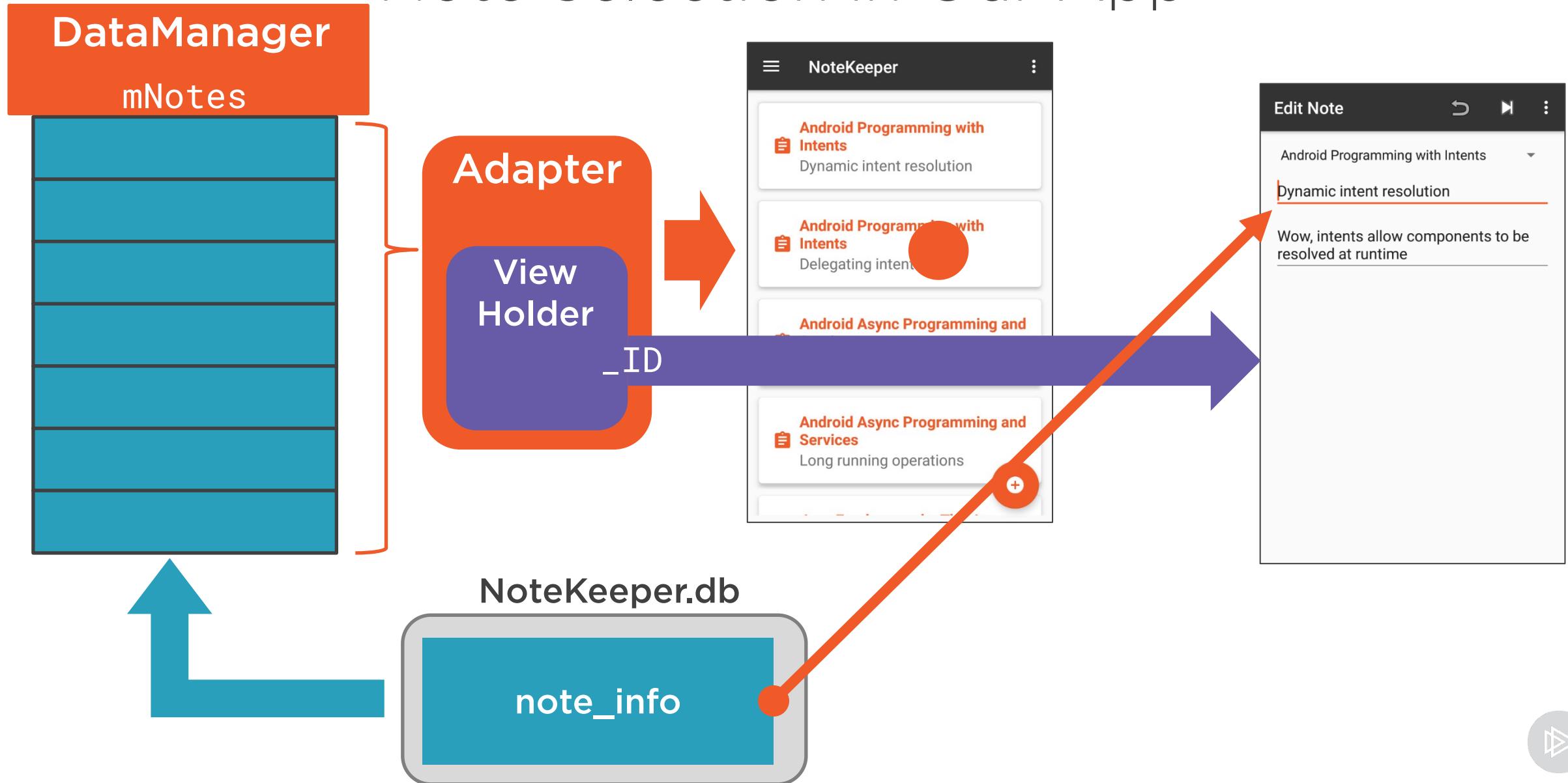
# Note Selection in Our App



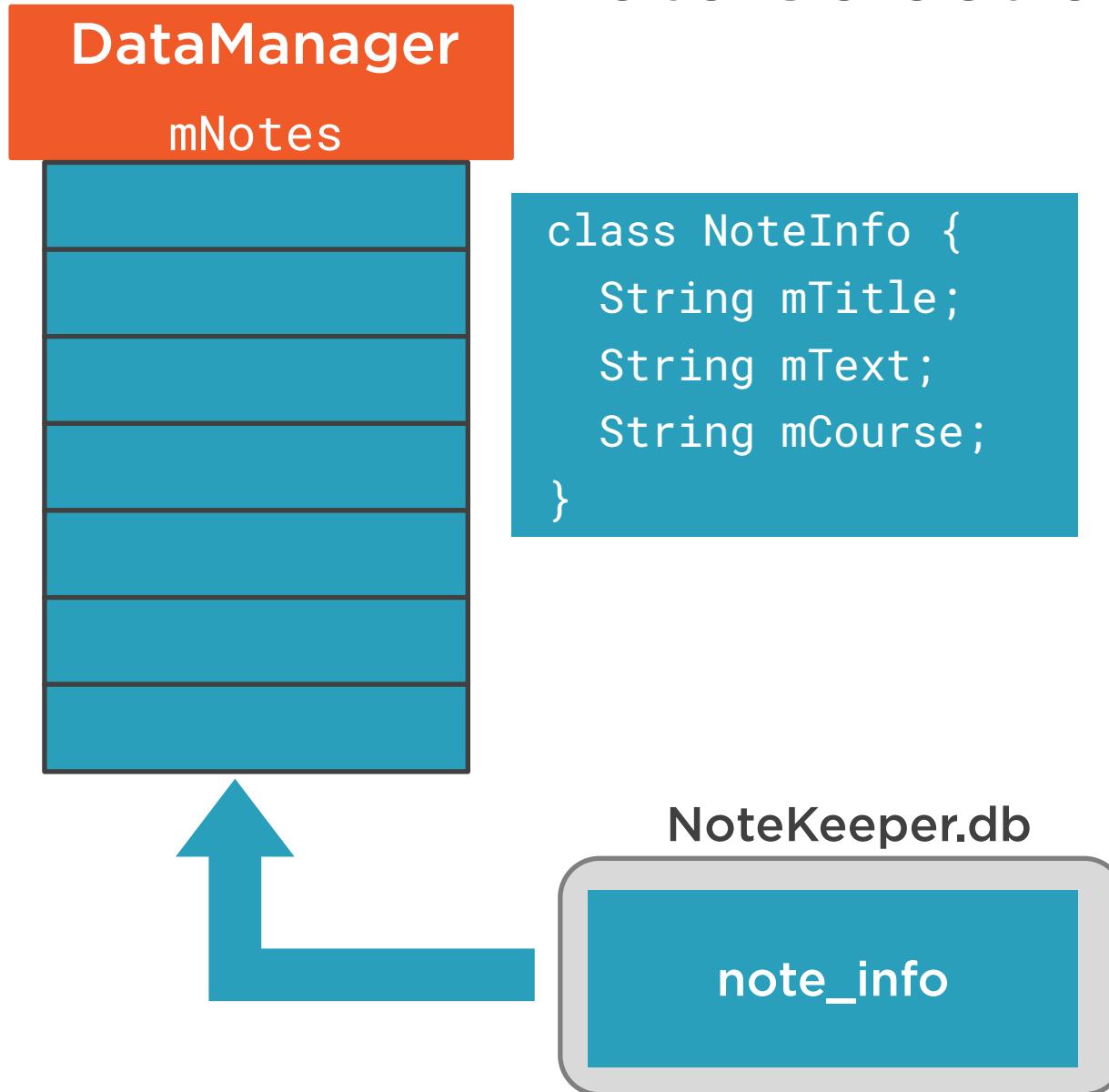
# Note Selection in Our App



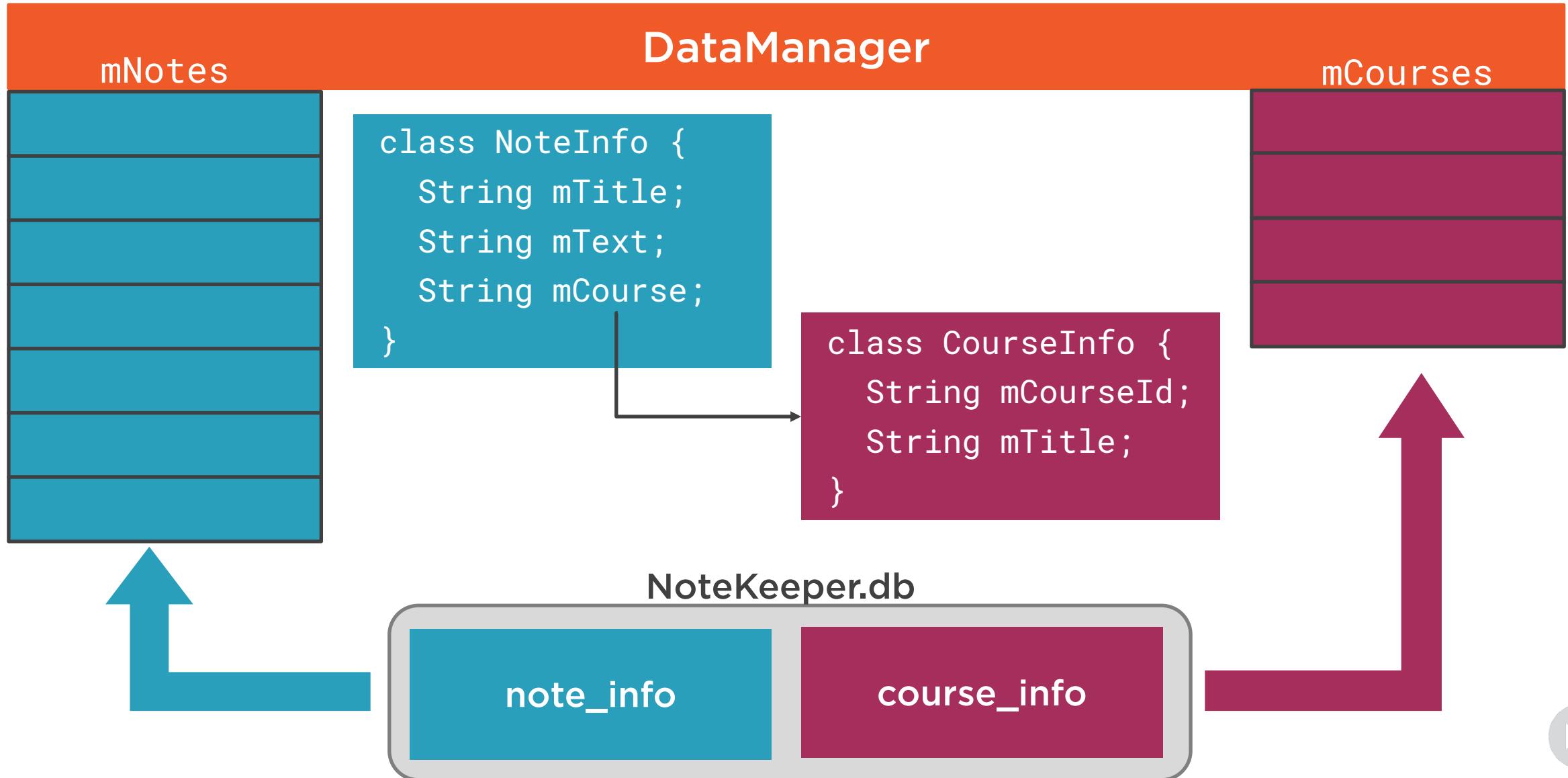
# Note Selection in Our App



# Note Selection in Our App



# Note Selection in Our App



# Accessing Data

**Data access has distinct phases**

- Request database connection
- Issue query
- Move through results



# Request Database Connection

## Connection provided by **SQLiteOpenHelper**

- Use your open helper implementation
- Call `getReadableDatabase`
- Returns `SQLiteDatabase` reference



# Issue Query

## **SQLiteDatabase query method**

- Parameters describe query details
- Unneeded parameters can be null

## **Fundamental query parameters**

- Table name
- Columns to be returned



# Move Through Results

## Cursor

- Provides access to query result
- Query result can have 0, 1, or more rows
- Result interaction occurs row-by-row



# Move Through Results

## Cursor maintains a current position

- Initially positioned before first row
- Must explicitly move to desired row

## Cursor.moveToNext

- Positions to next row in result
- Returns false when moves past end



# Moving Through Results

note\_info

_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Delegating intents	PendingIntents are powerful...	android_intents

query (



note\_title



course\_id

android\_intents

Dynamic intent resolution

java\_lang

Anonymous classes

android\_intents

Delegating intents



# Move Through Results

## **Cursor.moveToPrevious**

- Positions to previous row in result
- Returns false when moves past beginning



# Moving Through Results

note\_info

_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Delegating intents	PendingIntents are powerful...	android_intents

`query ("note_info", {"course_id", "note_title"}, ...);`



course\_id

note\_title

android_intents	Dynamic intent resolution
java_lang	Anonymous classes
android_intents	Delegating intents



# Move Through Results

## **Cursor.moveToFirst**

- Positions to first row in result

## **Cursor.moveToLast**

- Positions to last row in result

## **Cursor.moveToPosition**

- Positions to specified row in result
- Uses zero-based position index



# Moving Through Results

**Can access current row column values**

- Use typed Cursor.getXXX methods
- Use zero-based column position



# Moving Through Results

note\_info

_id	note_title	note_text	course_id
1	Dynamic intent resolution	Wow, intents allow components...	android_intents
2	Anonymous classes	Anonymous classes simplify...	java_lang
3	Delegating intents	PendingIntents are powerful...	android_intents

```
query ("note_info", {"course_id", "note_title"}, ...);
```

0

1

note1title

course\_id

android\_intents

java\_lang

android\_intents

Dynamic intent resolution

Anonymous classes

Delegating intents



# Moving Through Results

## Avoid hard-coding column positions

- Makes code fragile

## Request position of column name

- Cursor.getColumnIndex
- Accepts column name
- Returns column position in result



# Moving Through Results

## **Close Cursor when done with it**

- `Cursor.close`
- May leak system resources if not closed



# Ordering Results

**Can specify row order**

- Order by columns in result

**Passed as a string to query method**

- Pass column name



# Ordering Results

## Can sort by multiple columns

- List columns comma separated
- First column is primary sort
- Second column sorted within primary
- Third sorted within second and so on



# Ordering Results

**Can sort in descending order if desired**

- Follow column name by “ DESC”
- Applies only to that column



# SQLite

## Relational database

- Provides structured data storage

## Lightweight

- Runs in-process
- Optimized for common usage scenarios

## Portable

- Database stored as a file
- Broad operating system support



# Database Interaction

## Structured Query Language (SQL)

- Domain specific language
- English-like in nature

## SQL Challenges

- Parsing is resource intensive
- Potential for SQL based attacks



# Database Interaction

## Operation specific methods

- Preferable to SQL commands
- Help protect against SQL based attacks

## Apps use combination of SQL and methods

- Most data tasks can be done with methods
- Database management tends to need SQL
- Use methods whenever available



# Relational Structure

## Table

- A set of related entities
- Composed of columns and rows
- Generally 1 or more tables in a database



# Relational Structure

## Column

- An attribute of an entity
- Sometimes referred to as a field
- Generally 1 or more columns in a table

## Row

- An entity instance
- Sometimes referred to as record
- 0 or more rows in a table



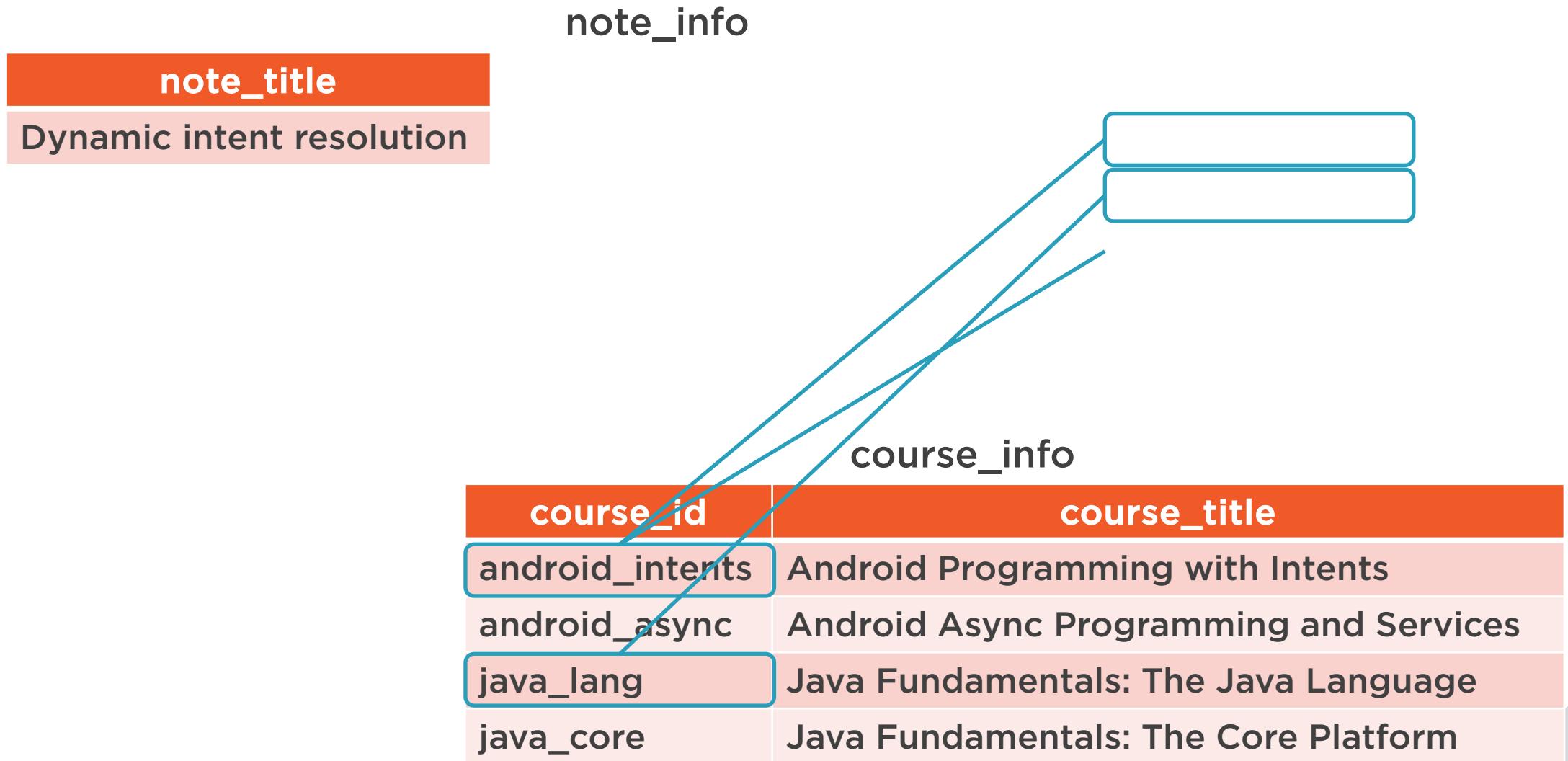
# Our Database

course\_info

course\_title



# Our Database



# Database Contract Class

**Holds information about database**

- Organization and structure (i.e. schema)
- Primarily constants describing DB
- Class is normally non-creatable



# Database Contract Class

## Nested class for each table

- Provides constant for table name
- Provides constants for column names
- May contain key SQL statements

## Table creation SQL

- Uses “CREATE TABLE” statement
- Names the table
- Describes the table columns



# Table Columns

## Describing table columns

- Name
- Storage class
- Constraints



## Storage Class

### Storage class is optional

- SQLite does not have rigid typing
- Any column can store any type
- Storage class influences storage affinity



## Storage Class

### Storage class is optional

- SQLite does not have rigid typing
- Any column can store any type
- Storage class influences storage affinity

	“123”	123	123.0	
				123.1 → 123.1



## Storage Class

### Storage class is optional

- SQLite does not have rigid typing
- Any column can store any type
- Storage class influences storage affinity

	“123”	123	123.0	“abc”
BLOB	“123”	123	123.0	
TEXT	“123”	“123”	“123.0”	
INTEGER	123	123	123	
REAL	123.0	123.0	123.0	
NUMERIC	123	123	123	



# Constraints

**Constraints restrict allowable content**

- Associate with columns as needed
- Automatically enforced by database



# Common Constraints

## NOT NULL

- Column cannot contain null

## UNIQUE

- No two rows can have the same value for the specified column



# Primary Key

## PRIMARY KEY

- Provides unambiguous row identity
- A table has no more than one



# Android Framework Friendly Tables

**SQLite tags each row with a unique integer**

- Automatically assigns the value
- Provides fast and efficient row access

**Create tables that work well with framework**

- Give table an integer primary key
- Primary key is associated with row tag
- Use BaseColumn.\_ID as column name



# Our Database

note\_info

note_title	note_text	course_id
Dynamic intent resolution	Wow, intents allow components...	android_intents
Anonymous classes	Anonymous classes simplify...	java_lang
Delegating intents	PendingIntents are powerful...	android_intents

course\_info

course_id	course_title
android_intents	Android Programming with Intents
android_async	Android Async Programming and Services
java_lang	Java Fundamentals: The Java Language
java_core	Java Fundamentals: The Core Platform



# Database Access and Creation

## Database access has implications

### Database creation

- Must check whether database exists
- DB must be created if doesn't exist

### Database versioning

- App DB structure may change over time
- Must track current DB version
- App must indicate expected DB version
- DB must be updated when necessary



# Database Open Helper

## **SQLiteOpenHelper**

- Provides DB access methods
- Manages DB creation and versioning

## **Extend to meet your app's needs**

- Provide expected database version
- Provide database filename
- Override appropriate methods



# Database Open Helper

## Override onCreate

- Called if database doesn't exist
- Execute SQL to create tables
- Add initial data if necessary

## Override onUpgrade

- Called when newer DB version expected
- Execute SQL to upgrade tables
- Preserve existing data if necessary



# Database Access and Creation

## **Using your database open helper class**

- Instantiate in activity's onCreate
- Maintain reference as member field
- Close in activity's onDestroy



# Database Access and Creation

**Helper class provides DB access methods**

- getReadableDatabase
- getWritableDatabase
- Trigger creation/version checks as needed
- Return SQLiteDatabase reference

**SQLiteDatabase**

- Provides database interaction methods
- Operation specific methods
- Execute SQL statements with execSQL



# Database Access

## Provided by your helper implementation

- Create/version checks occur on 1<sup>st</sup> access
- Calls onCreate/onUpgrade as appropriate

## Using your helper implementation

- Instantiate in activity's onCreate
- Maintain reference as member field
- Close in activity's onDestroy



# Database Access

**Helper class provides DB access methods**

- getReadableDatabase
- getWritableDatabase
- Return SQLiteDatabase reference

**SQLiteDatabase**

- Provides database interaction methods
- Operation specific methods
- Execute SQL statements with execSQL



# Database Open Helper

## **SQLiteOpenHelper**

- Manages DB creation and versioning
- Extend to customize implementation

## **Implement constructor**

- Call super class constructor
- Pass expected database version
- Pass database filename



# Database Access and Creation

**Database must be created before first use**

- Avoid creating during app launch

**Database versioning**

- App DB structure may change over time
- DB must track current version
- App must indicate expected DB version
- DB must be updated when necessary



# SQLiteDatabase

**Provides database interaction methods**

- Operation specific methods
- Execute SQL statements with execSQL

**Database access must be handled appro**



# Android Framework Friendly Tables

## **Create tables that work well with framework**

- Include column with unique integer value
- Use `BaseColumn._ID` as column name



# Creating a Database Table

## **Storage class is optional**

- SQLite does not have rigid typing
- Any column can store any type
- Class influences storage affinity



# Database Contract Class

## Nested class for each table

- Provides constant for table name
- Provides constants for column names
- Implements BaseColumns
  - Provides \_ID column name constant
- May contain key SQL statements



# Creating Database Table

## Use SQL

- “Create Table” statement

## Describe columns

- Name
- Storage class
- Constraints



# Our Tables

note\_info

_id	note_title	note_text	course_id
0	Dynamic intent resolution	Wow, intents allow components...	android_intents
1	Anonymous classes	Anonymous classes simplify...	java_lang
2	Delegating intents	PendingIntents are powerful...	android_intents

course\_info

_id	course_id	course_title
0	android_intents	Android Programming with Intents
1	android_async	Android Async Programming and Services
2	java_lang	Java Fundamentals: The Java Language
3	java_core	Java Fundamentals: The Core Platform



# Creating a Database Table

## Storage class is optional

- SQLite does not have rigid typing
- Any column can store any type
- Storage class influences storage affinity

	“123”	123	123.0
<b>BLOB</b>	“123”	123	123.0
<b>TEXT</b>	“123”	“123”	“123”
<b>INTEGER</b>	123	123	123
<b>REAL</b>	123.0	123.0	123.0
<b>NUMERIC</b>	123	123	123



# Creating a Database Table

## Storage class is optional

- SQLite does not have rigid typing
- Any column can store any type
- Class influences storage affinity

## Available storage classes

- BLOB
- TEXT
- INTEGER
- REAL
- NUMERIC



# SQLite

## Relational database

- Provides structured data storage
- Structured Query Language (SQL)

## Lightweight

- Runs in-process
- Optimized for common usage scenarios

## Portable

- Database stored as a file
- Broad operating system support



# Relational Structure

## Table

- A set of related entities
- Composed of rows and columns
- Generally 1 or more tables in a database



# Relational Structure

## Row

- An entity instance
- Sometimes referred to as record
- 0 or more rows in a table

## Column

- An attribute of an entity
- Sometimes referred to as a field
- Can optionally allow null
- Generally 1 or more columns in a table



# Data Interaction

## Structured Query Language (SQL)

- Data interaction language
- English-like in nature

## Data interaction methods

- Available for most common operations
- Preferable to SQL commands
- Protect against SQL-oriented attacks



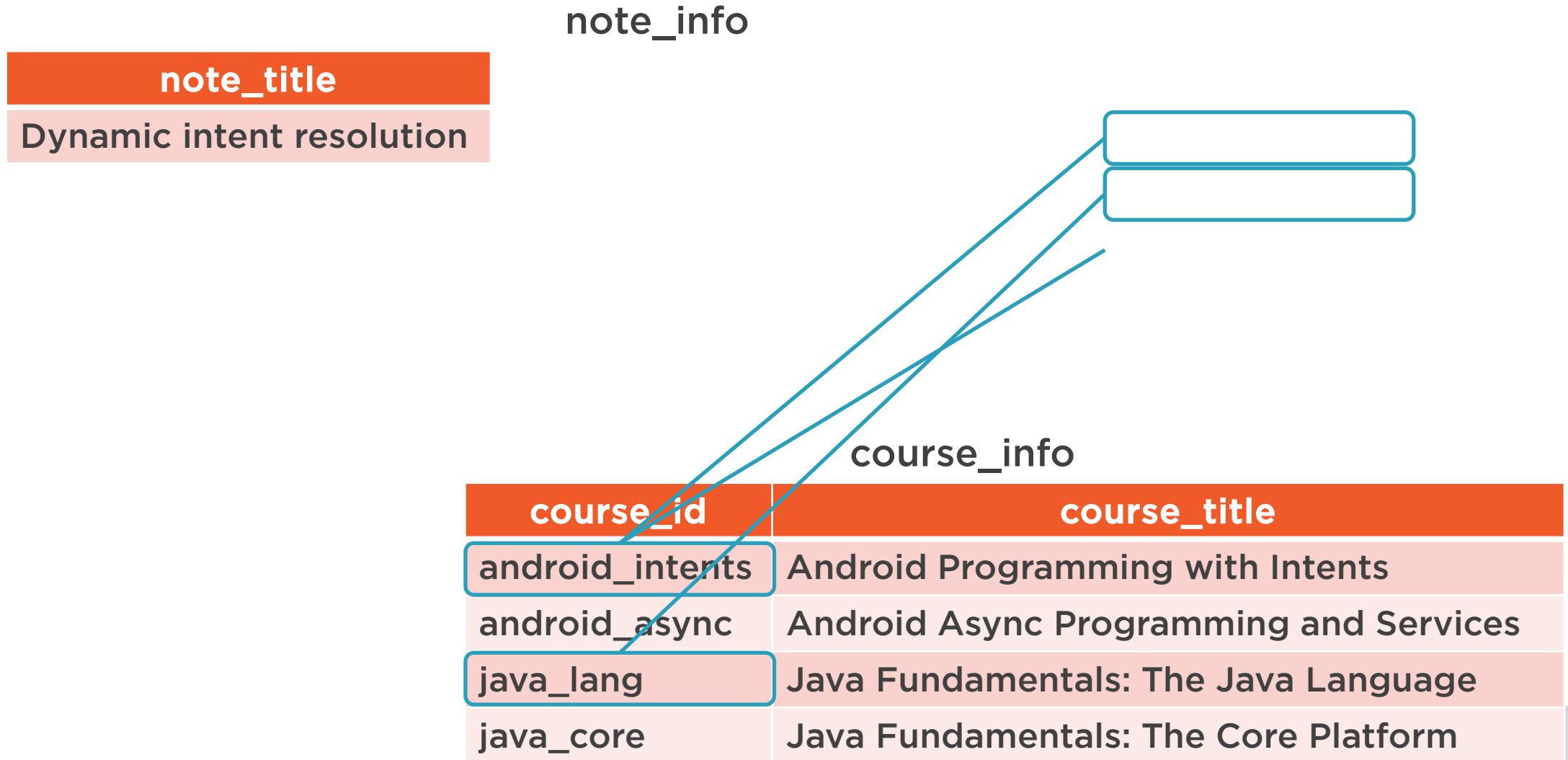
# Our Tables

course\_info

course\_title



# Our Tables



# Primary Key

**Column providing unambiguous row identity**

- Value is unique to each row
- Value is non-null

**SQLite tags each row with a unique integer**

- Will automatically assign the value
- Provides fast and efficient row access

**Create tables that work well with framework**

- Give table an integer primary key
- Use BaseColumn.\_ID as column name



# Our Tables

`note_info`

<code>note_title</code>	<code>note_text</code>	<code>course_id</code>
Dynamic intent resolution	Wow, intents allow components...	android_intents
Anonymous classes	Anonymous classes simplify...	java_lang
Delegating intents	PendingIntents are powerful...	android_intents

`course_info`

<code>course_id</code>	<code>course_title</code>
android_intents	Android Programming with Intents
android_async	Android Async Programming and Services
java_lang	Java Fundamentals: The Java Language
java_core	Java Fundamentals: The Core Platform



# Database and Table Creation

## Database contract class

- Describe tables and their columns
- Commands for creating tables

## Database access and creation

- Using SQLiteOpenHelper



# Retrieving Data

## Selecting data from a table

- Specifying result columns
- Parameterized row filtering
- Ordering results

## Handling results

- Explicit handling with a Cursor
- Connecting to views with CursorAdapter

## Lifecycle aware loading

- Maintaining a responsive UI with a Loader



# Modifying Data

## Inserting rows

- Specifying columns with ContentValues

## Deleting rows

- Parameterized row filtering

## Updating rows

- Specifying columns with ContentValues
- Parameterized row filtering



# Our Tables

note\_info

_id	note_title	note_text	course_id
0	Dynamic intent resolution	Wow, intents allow components...	android_intents
1	Anonymous classes	Anonymous classes simplify...	java_lang
2	Delegating intents	PendingIntents are powerful...	android_intents

course\_info

_id	course_id	course_title
0	android_intents	Android Programming with Intents
1	android_async	Android Async Programming and Services
2	java_lang	Java Fundamentals: The Java Language
3	java_core	Java Fundamentals: The Core Platform



# Our Tables

note_title	note_text	course_id
Dynamic intent resolution	Wow, intents allow components...	android_intents
Anonymous classes	Anonymous classes simplify...	java_lang
Delegating intents	PendingIntents are powerful...	android_intents

course_id	course_title
android_intents	Android Programming with Intents
android_async	Android Async Programming and Services
java_lang	Java Fundamentals: The Java Language
java_core	Java Fundamentals: The Core Platform



# Improving Note List Appearance

NoteKeeper

android\_intents|Dynamic intent resolution|Wow, intents allow components to be resolved at runtime

android\_intents|Delegating intents|PendingIntents are powerful; they delegate much more than just a component invocation

android\_async|Service default threads|Did you know that by default an Android Service will tie up the UI thread?

android\_async|Long running operations|Foreground Services can be tied to a notification icon

java\_lang|Parameters|Leverage variable-length parameter lists

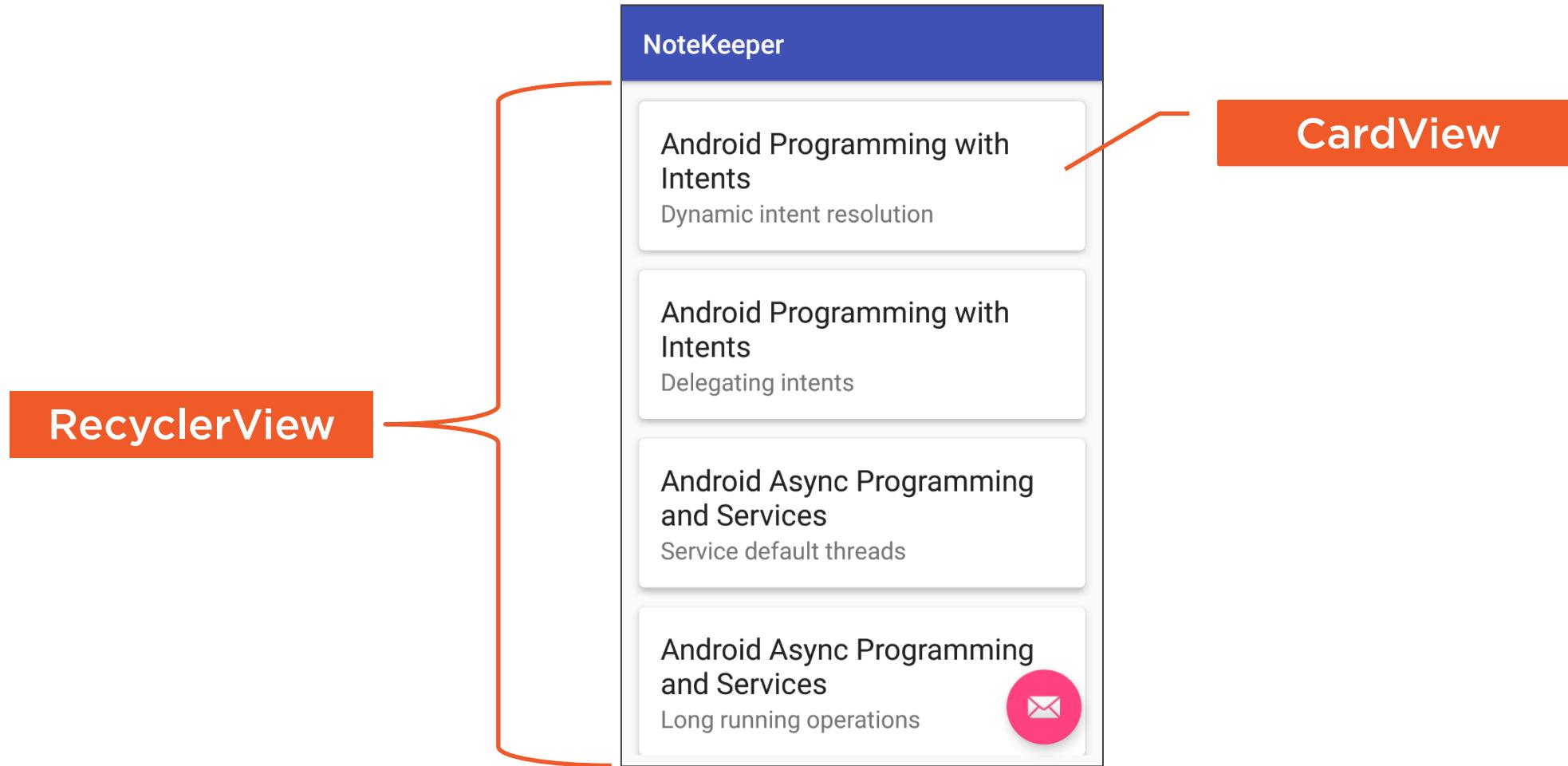
java\_lang|Anonymous classes|Anonymous classes simplify implementing one-use types

java\_core|Compiler options|The -jar option isn't compatible with the -cp option

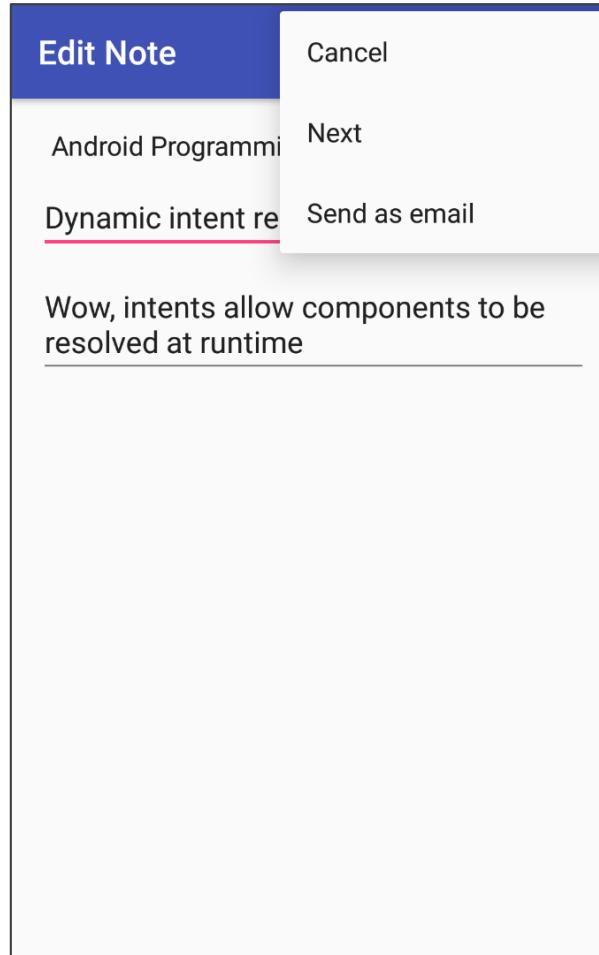
java\_core|Serialization|Remember to include serialVersionUID to assure version compatibility



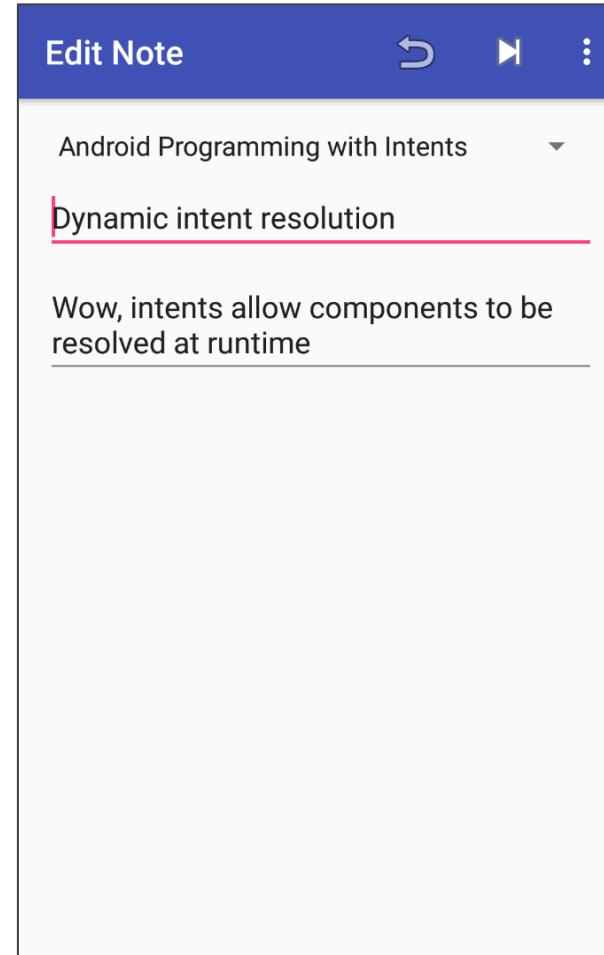
# Improving Note List Appearance



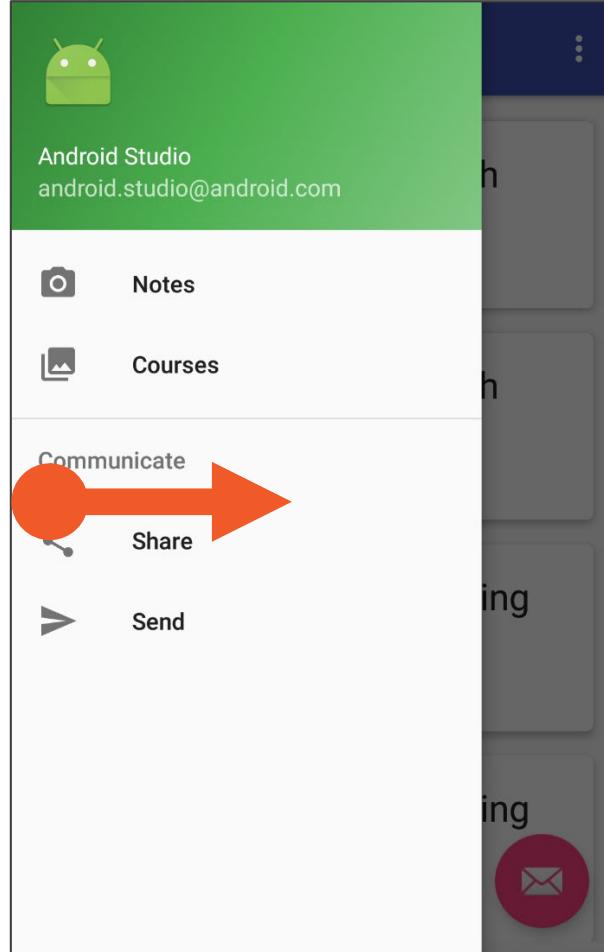
# Improving Note Interaction



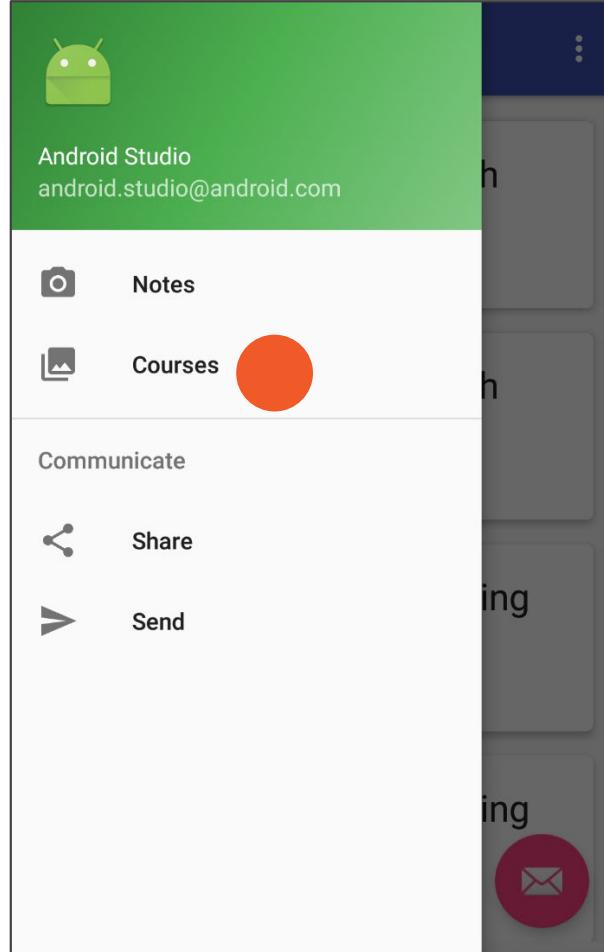
# Improving Note Interaction



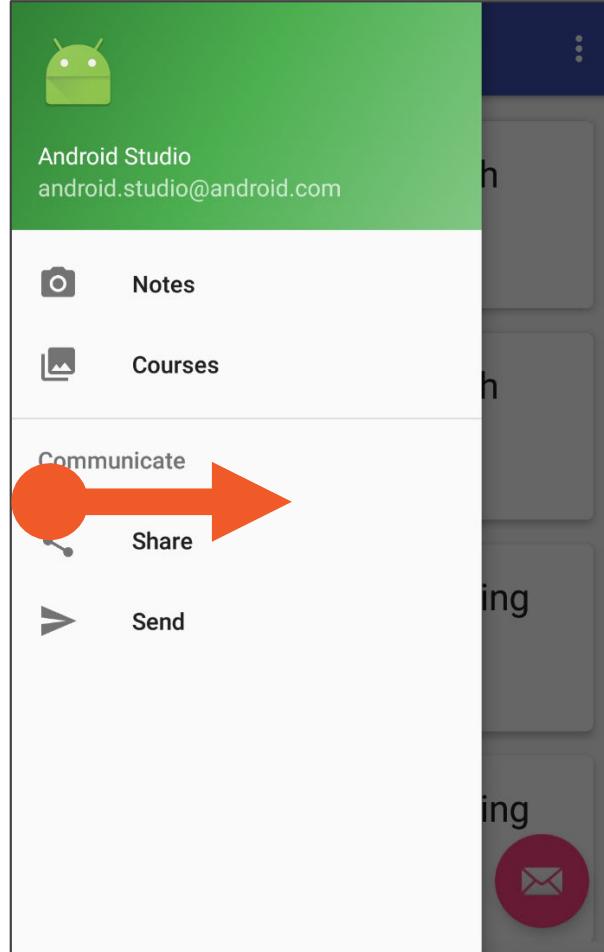
# Expanding Navigation



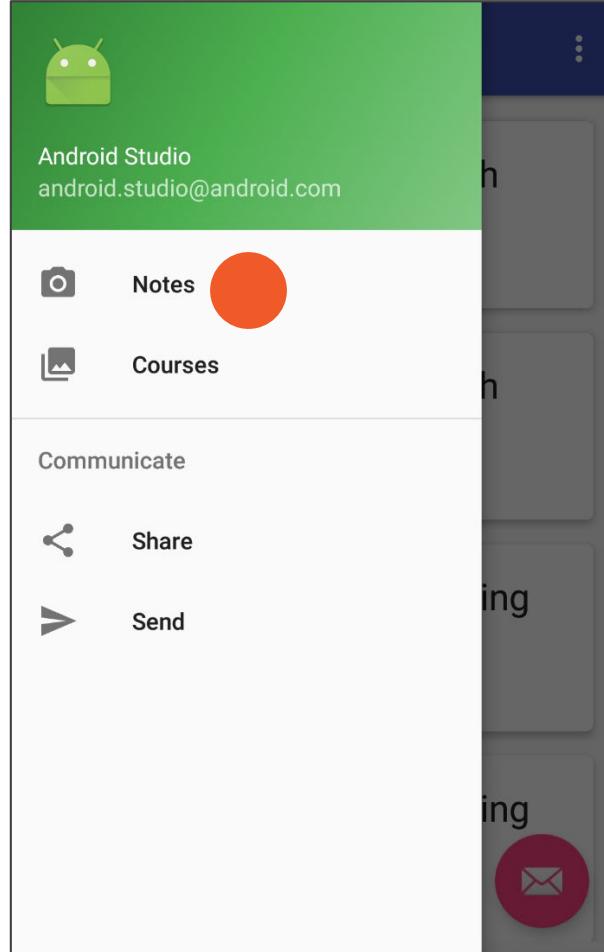
# Expanding Navigation



# Expanding Navigation

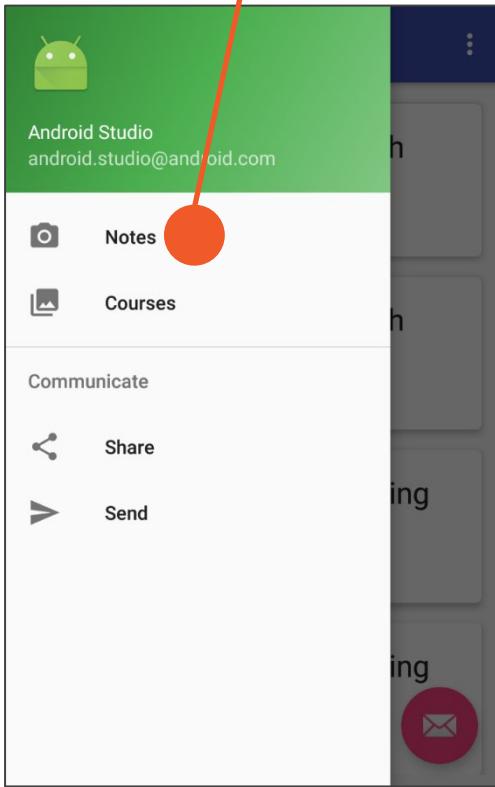


# Expanding Navigation

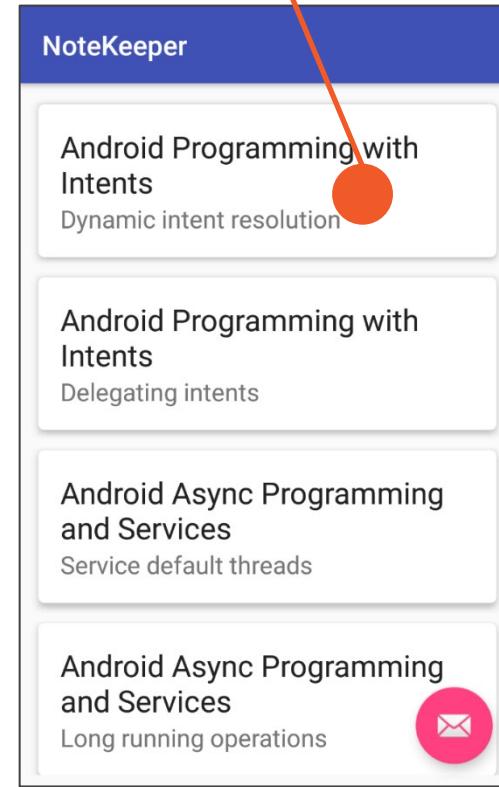


# Automated UI Testing

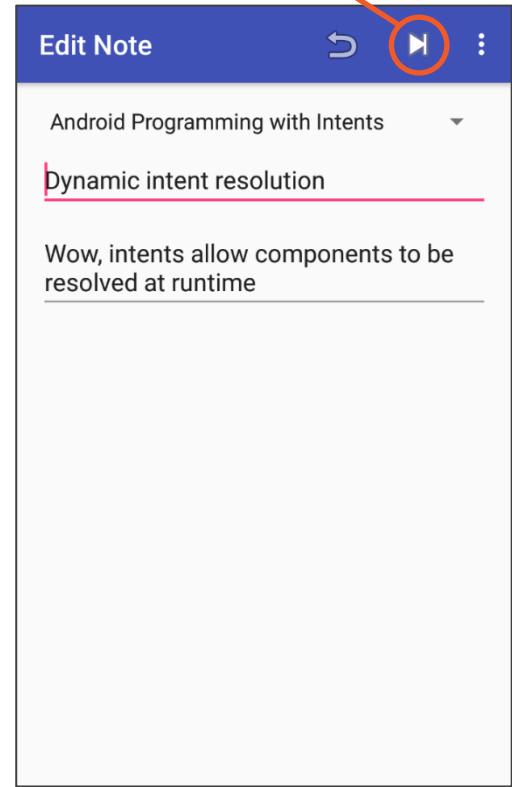
Navigation  
Drawer



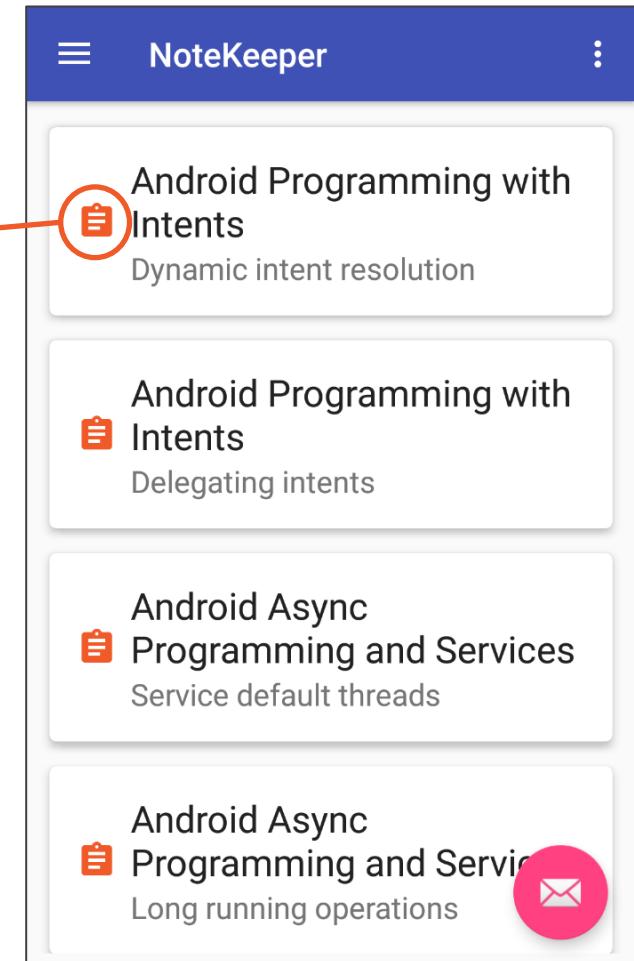
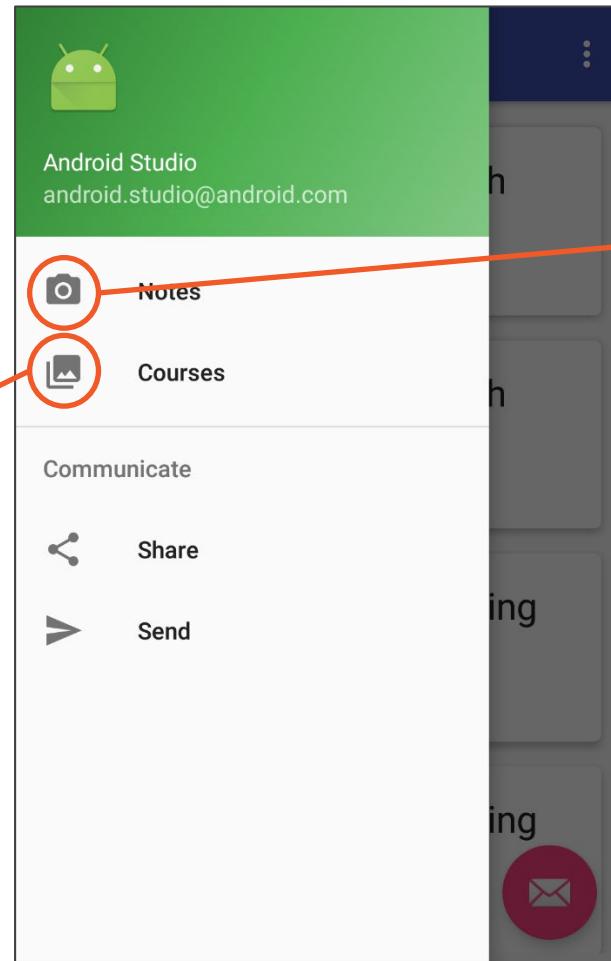
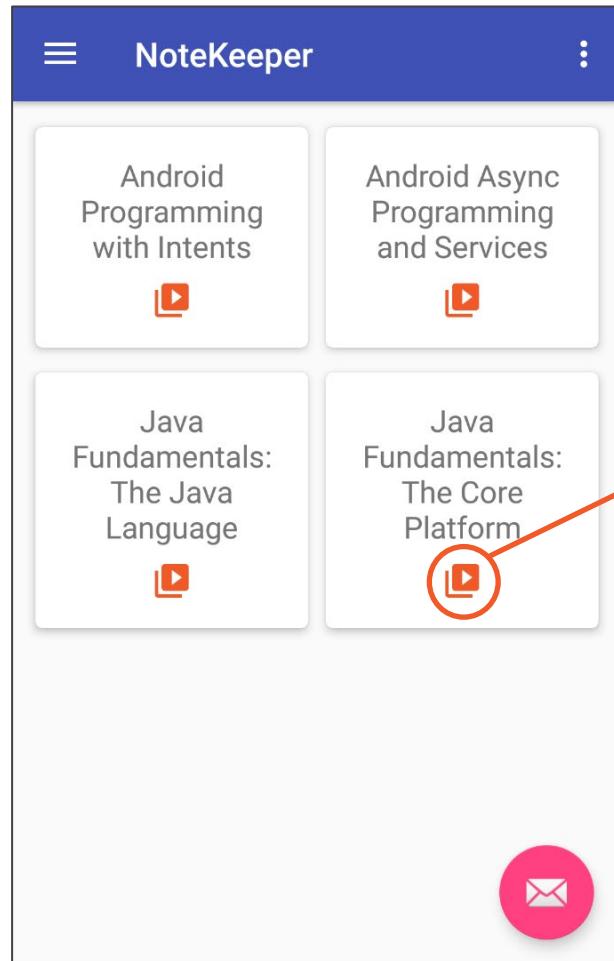
RecyclerView



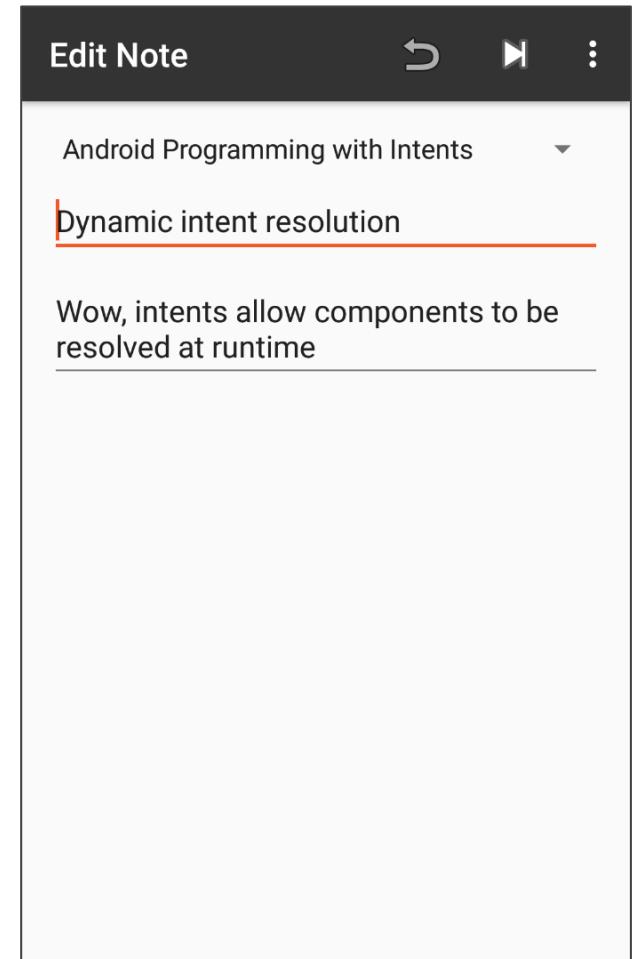
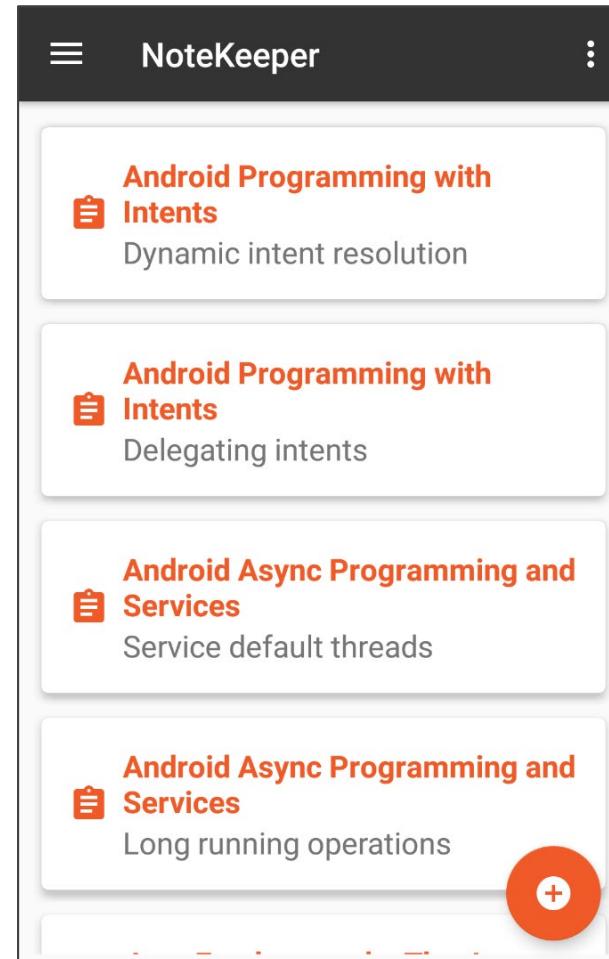
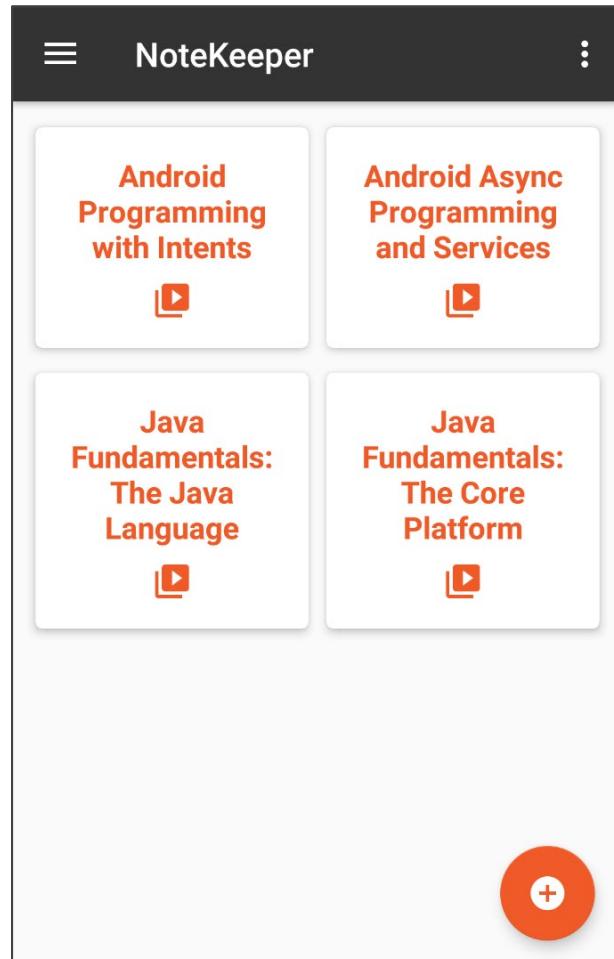
OptionsMenu



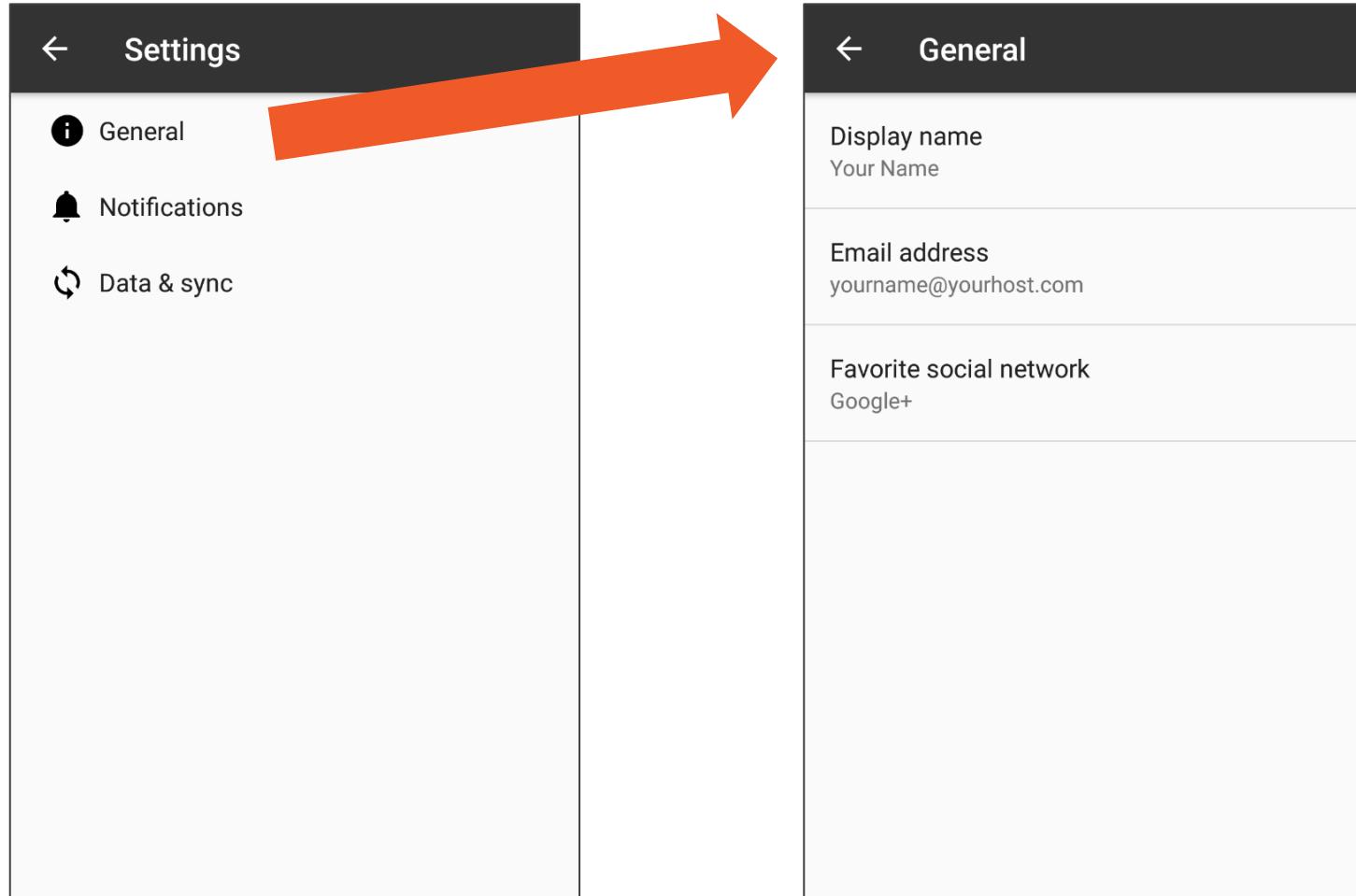
# Enriching App Appearance



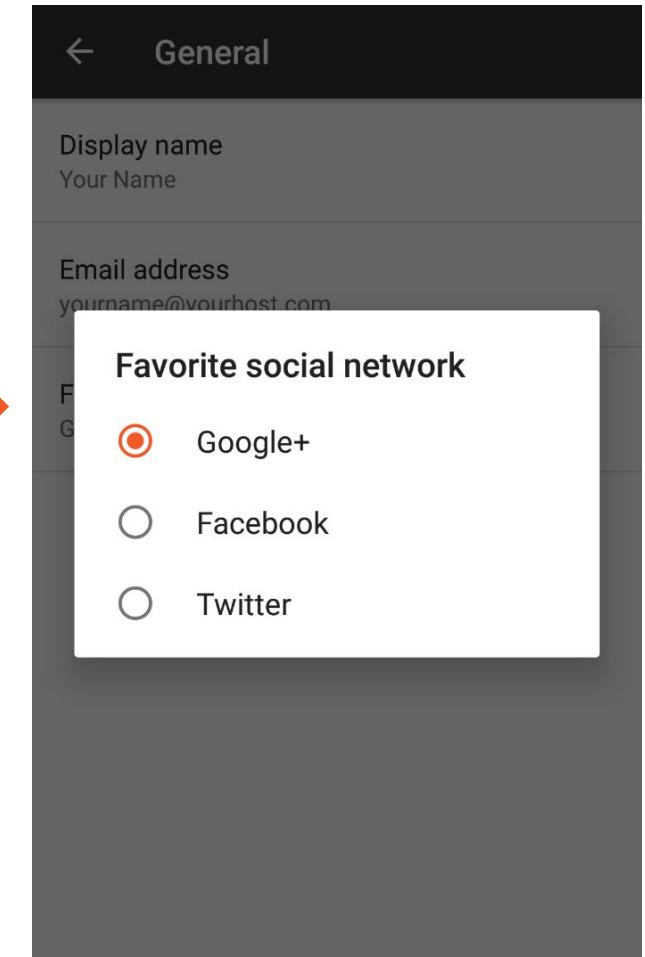
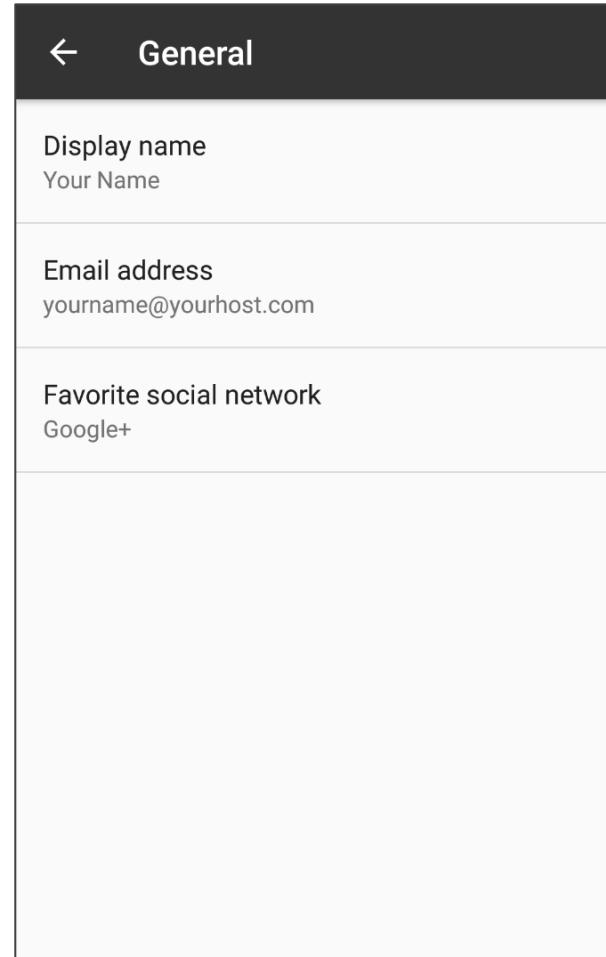
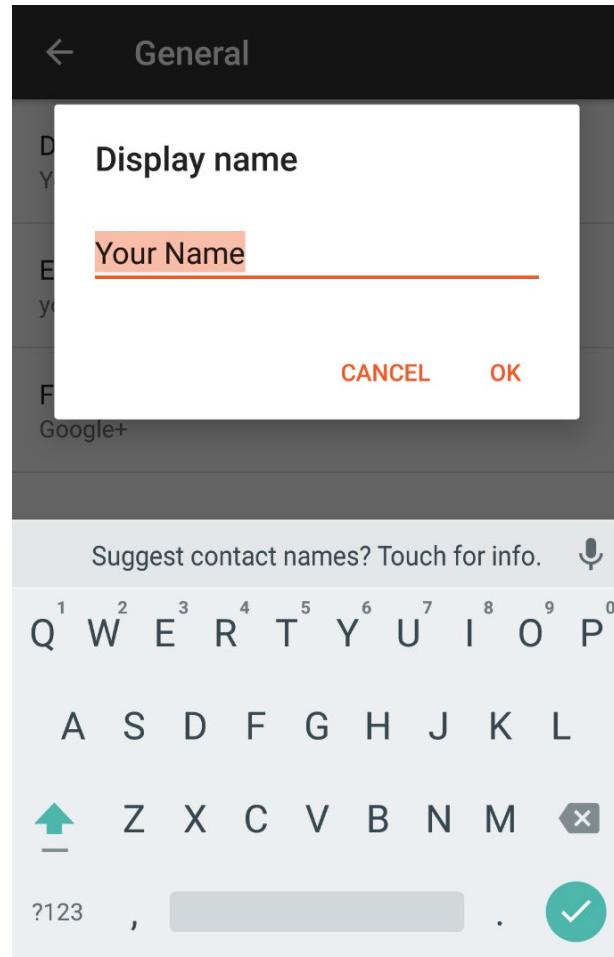
# Enriching App Appearance



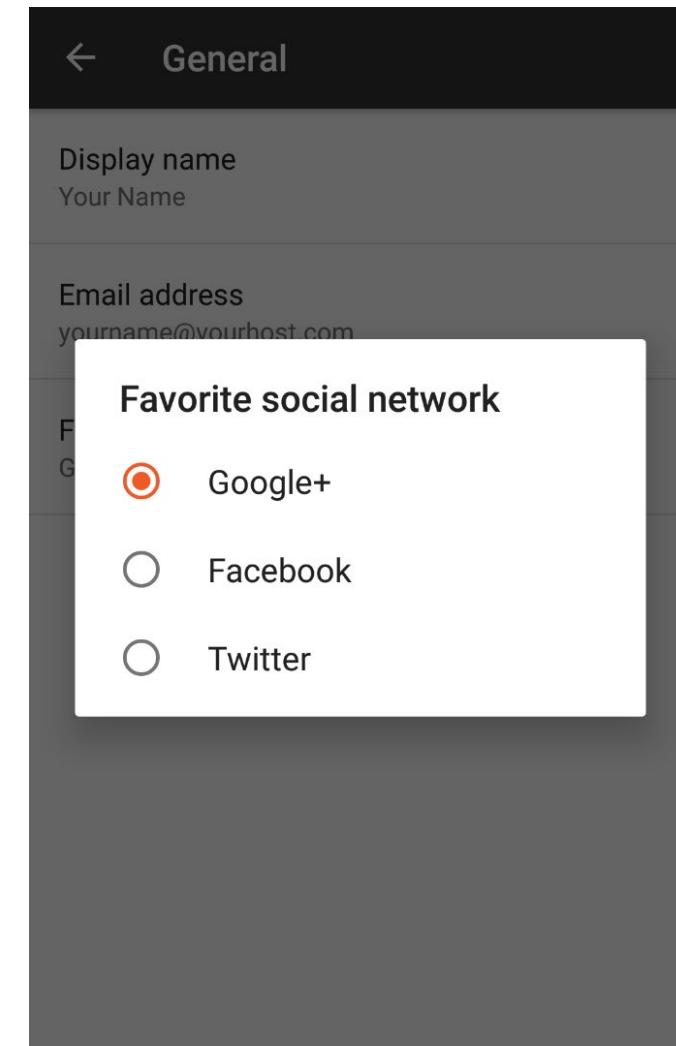
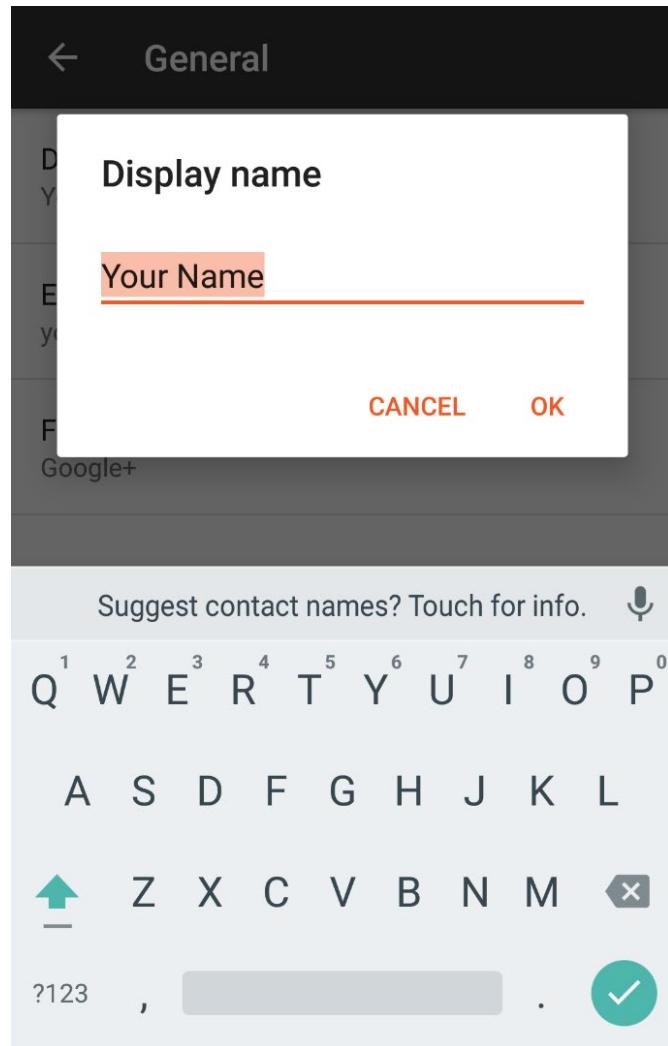
# Customizable Content and Behavior



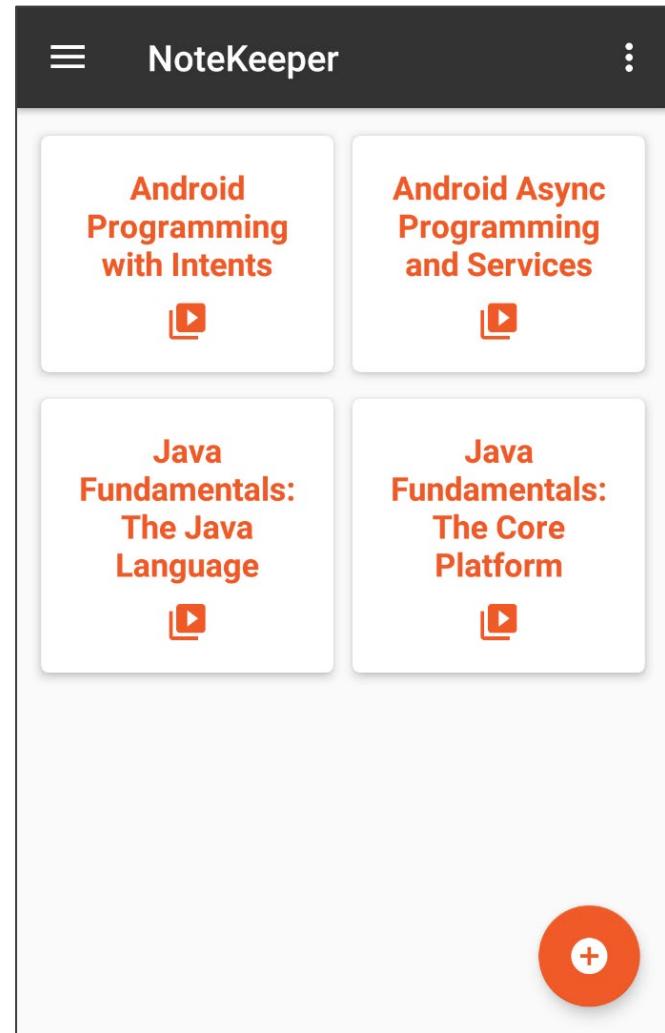
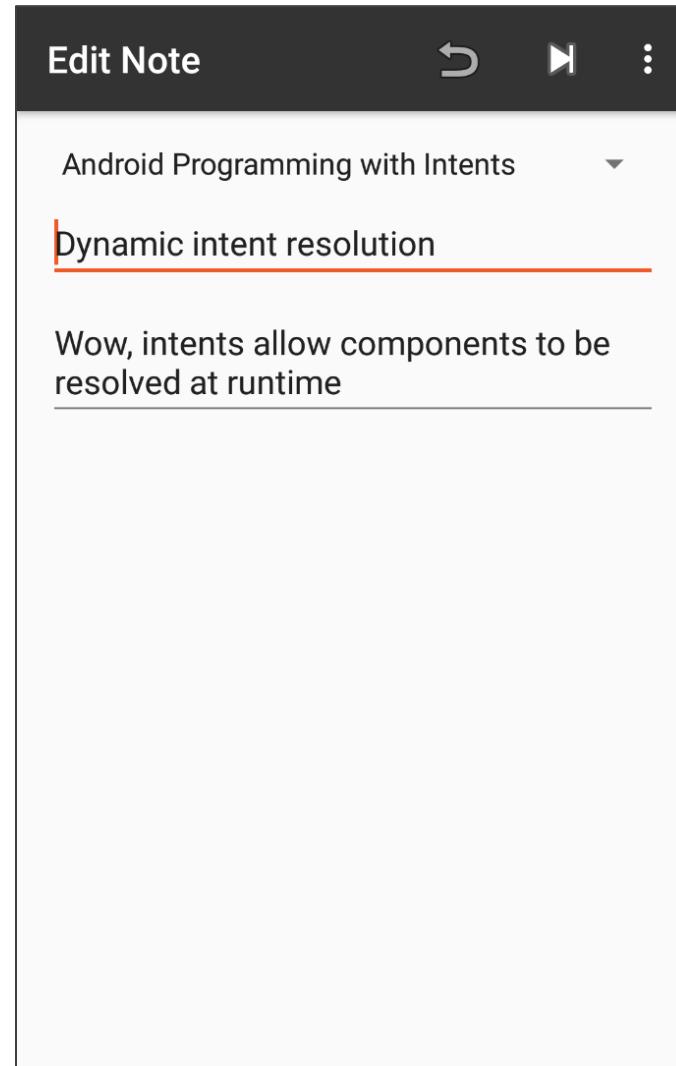
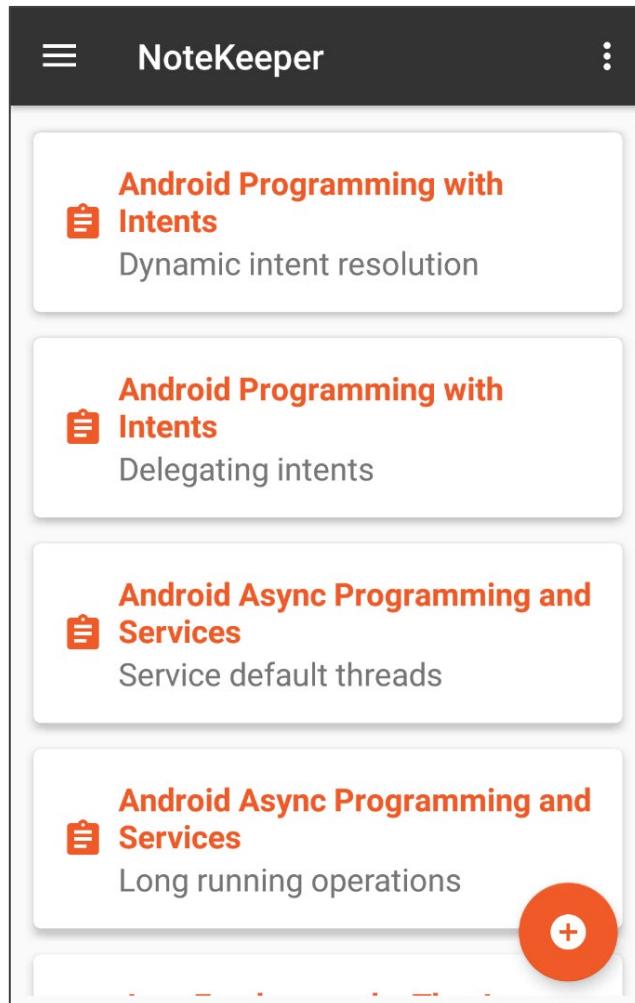
# Customizable Content and Behavior



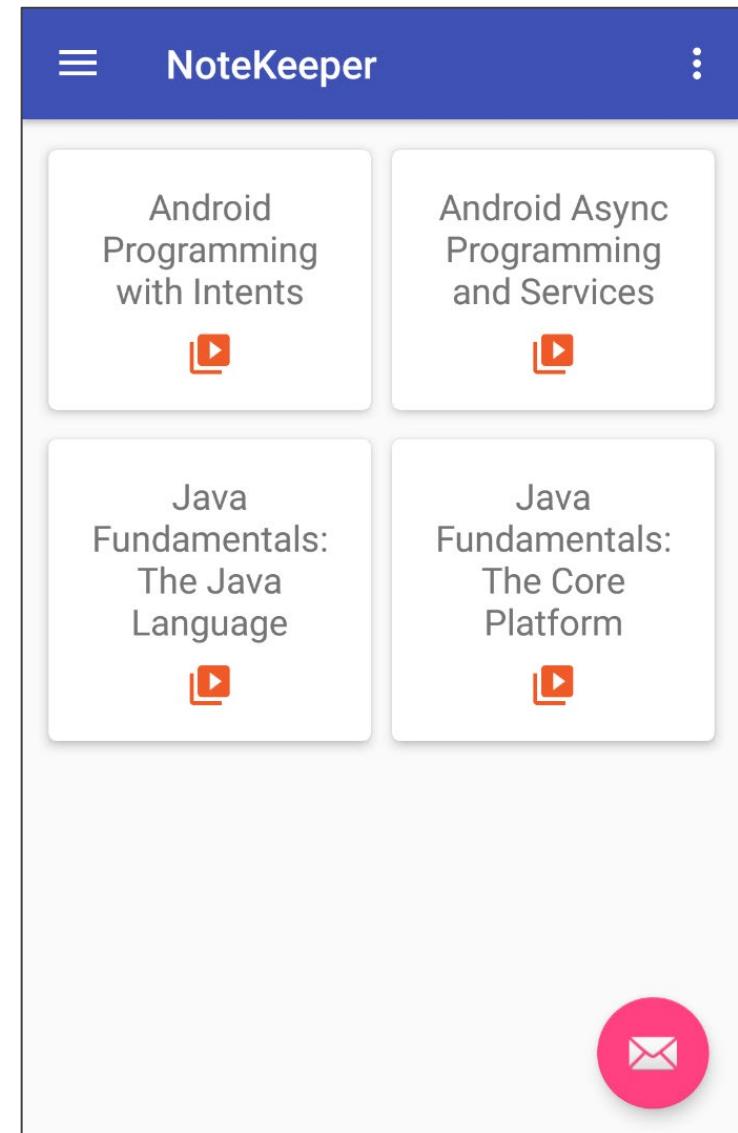
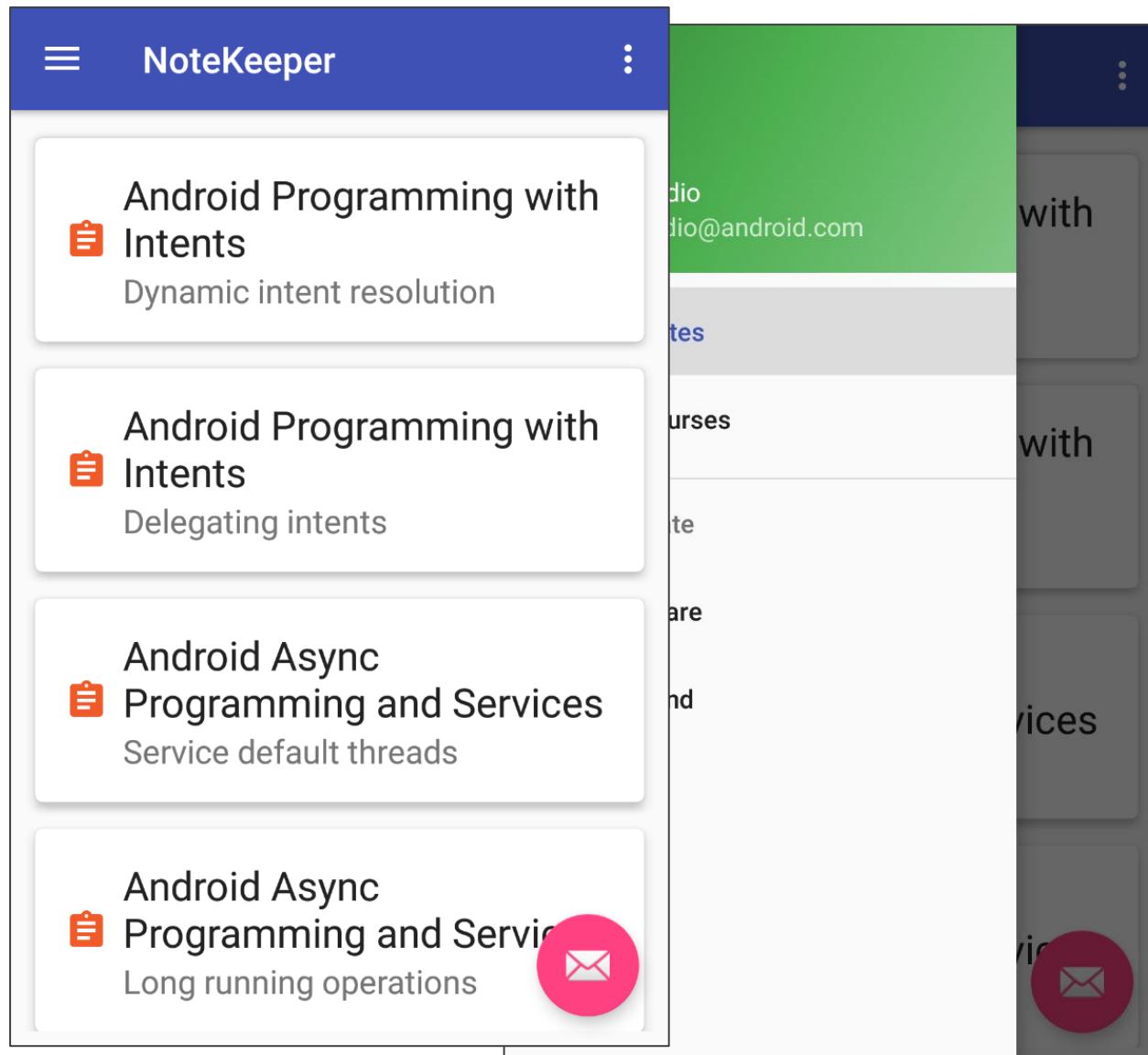
# Personalizing Content and Behavior



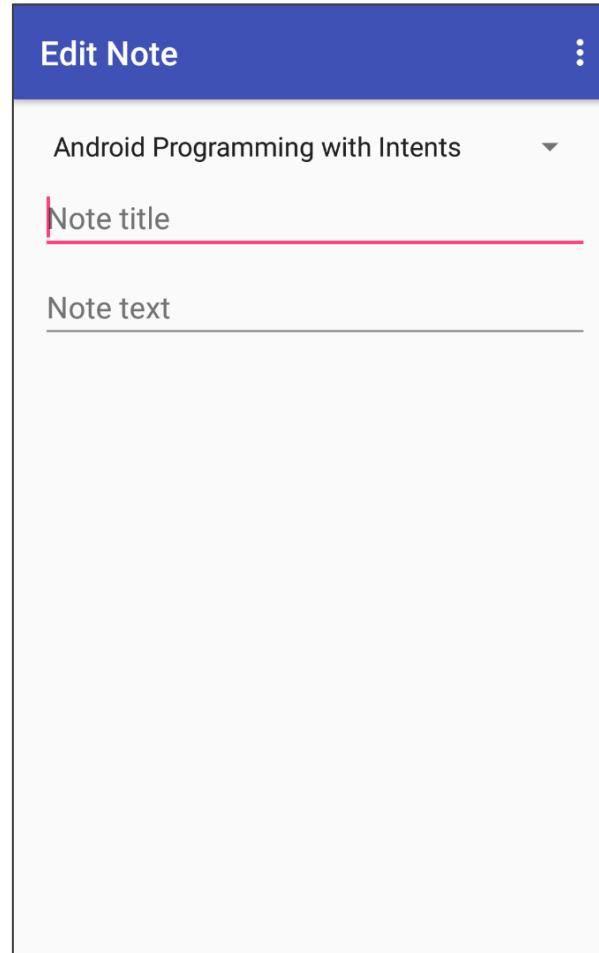
# Providing Consistency and Branding



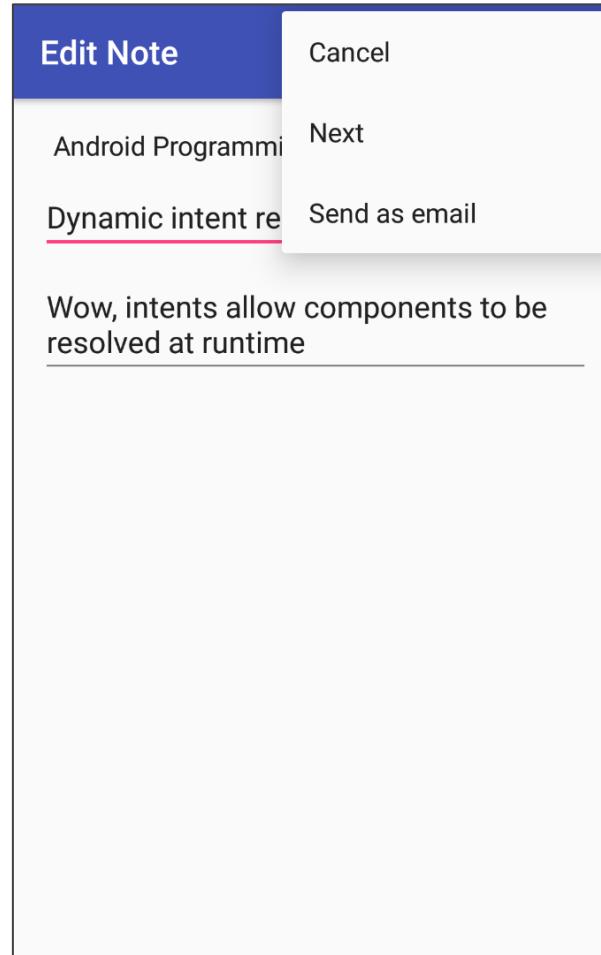
# Enriching App Appearance



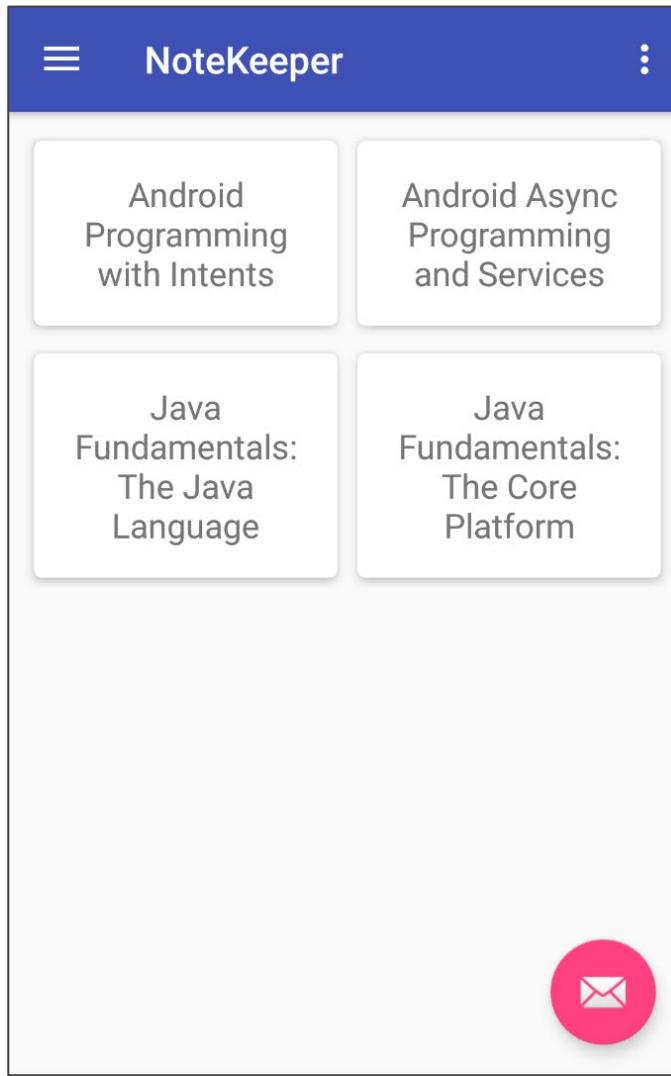
# Improving Note Interaction



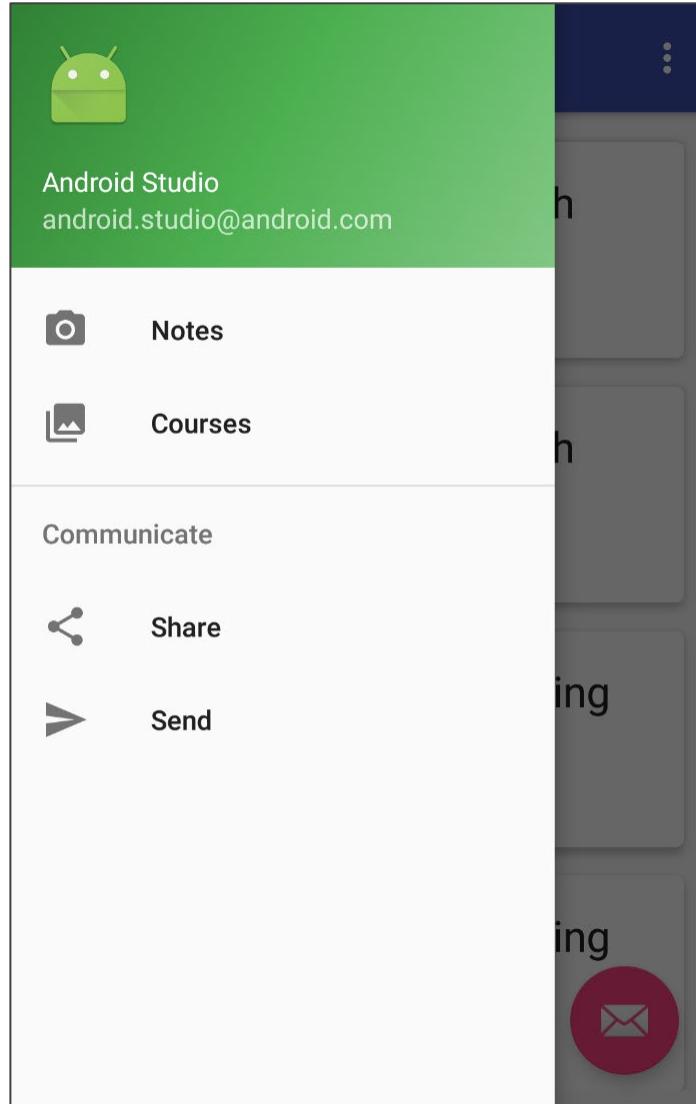
# Improving Note Interaction



# Expanding Navigation



# Expanding Navigation



# Settings Screens

**Apps generally have settings screens**

- Provide user customizable preferences
- Implementing follows a standard model



# Settings Screens

## Preference organization

- Group related preferences together

## Preference presentation

- Display preference title
- Display preference description
- Allow user to modify preference value

## Preference storage

- Store as name/value pair
- Provide default value



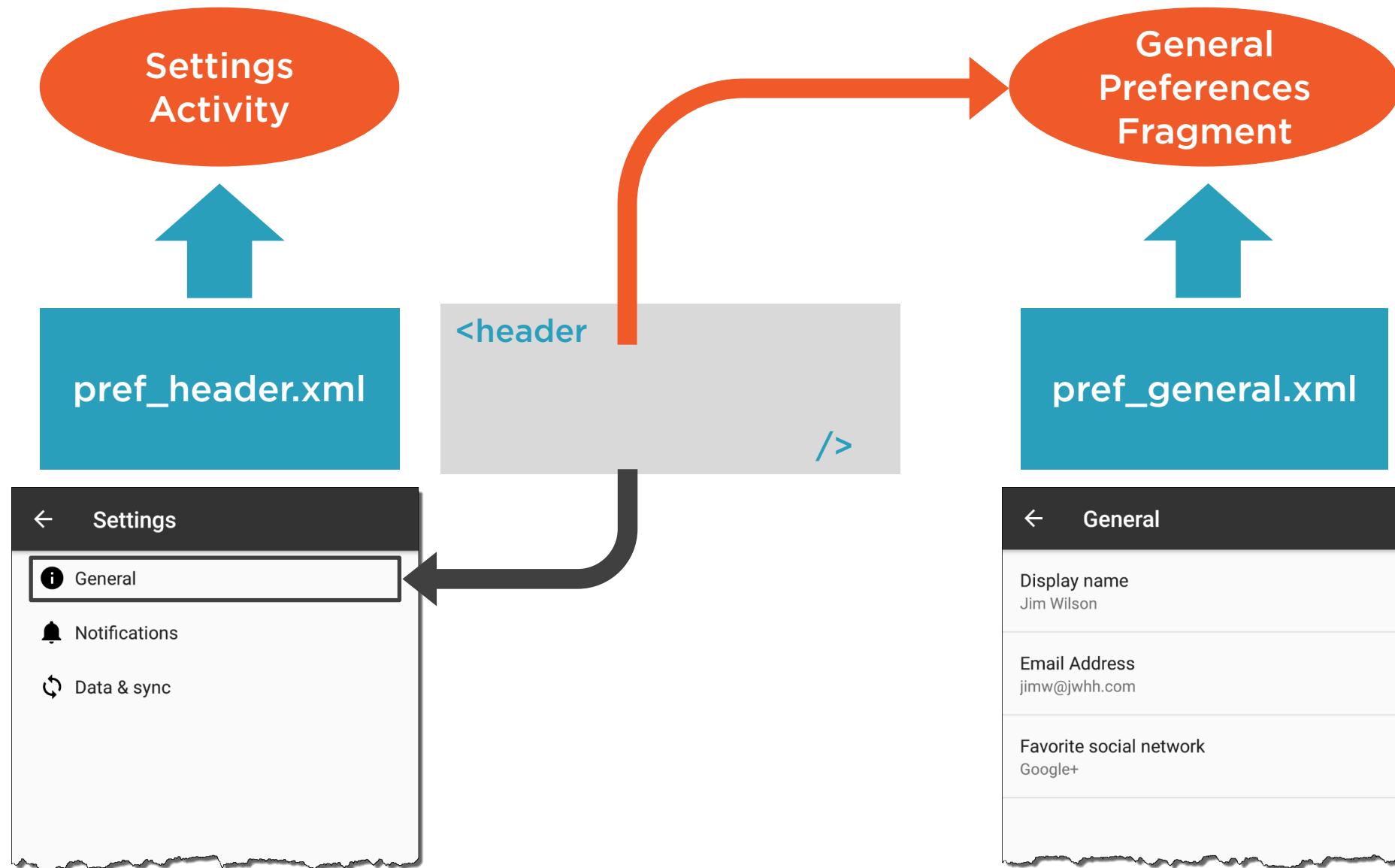
# Settings Screens

## Preference system

- Simplifies implementation
- Preferences described declaratively
- Settings-oriented UI classes
- Name/value pair storage management



# Settings Screens Resources and Classes



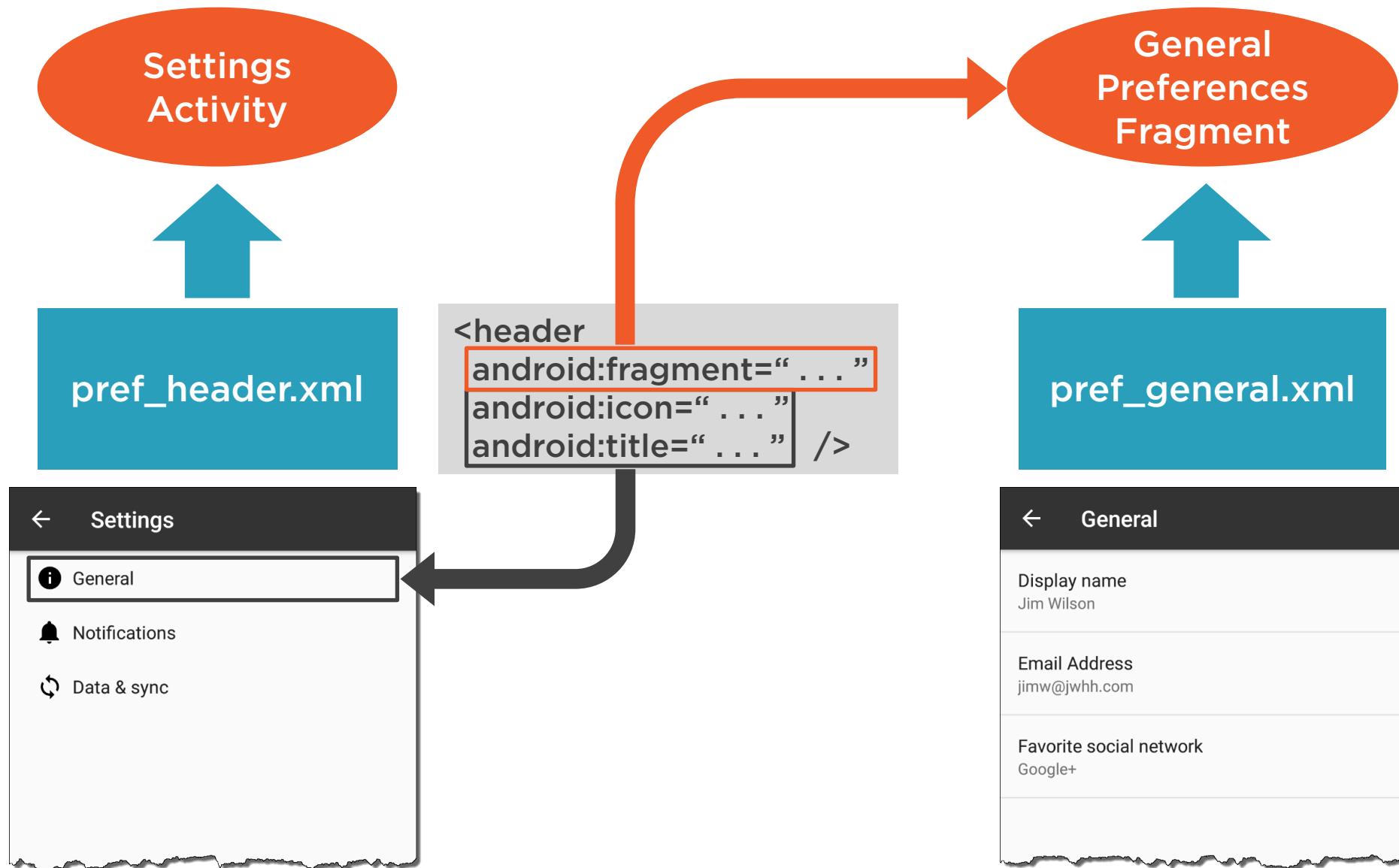
# Settings Screens Resources and Classes

## Preference-derived classes handle UI details

- Manages storage
- Presents title and summary
- Provides appropriate UI experience
  - UI experience driven by type



# Settings Resources and Classes



# View Styles

## **View appearance controlled by properties**

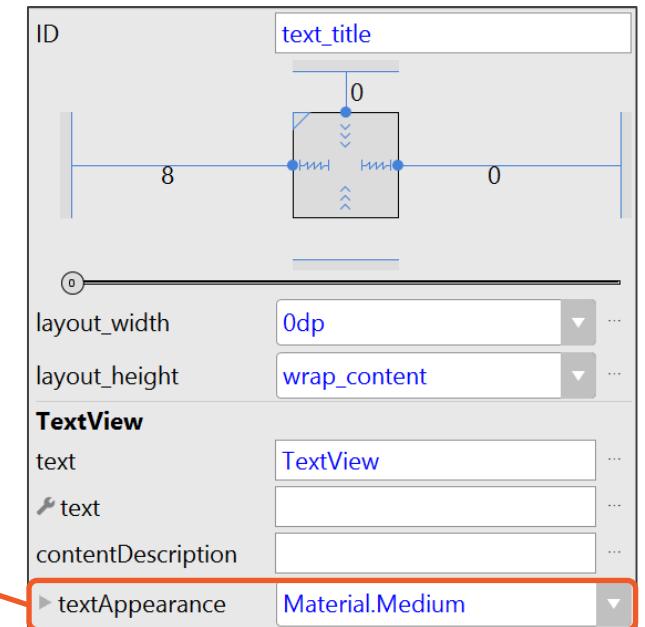
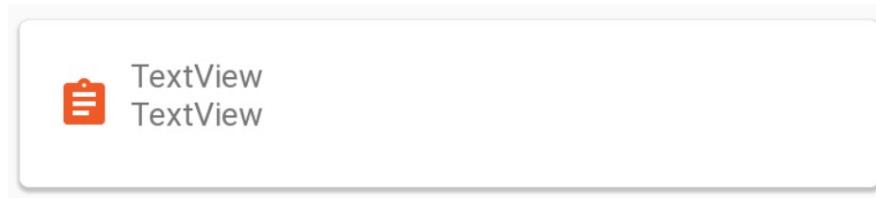
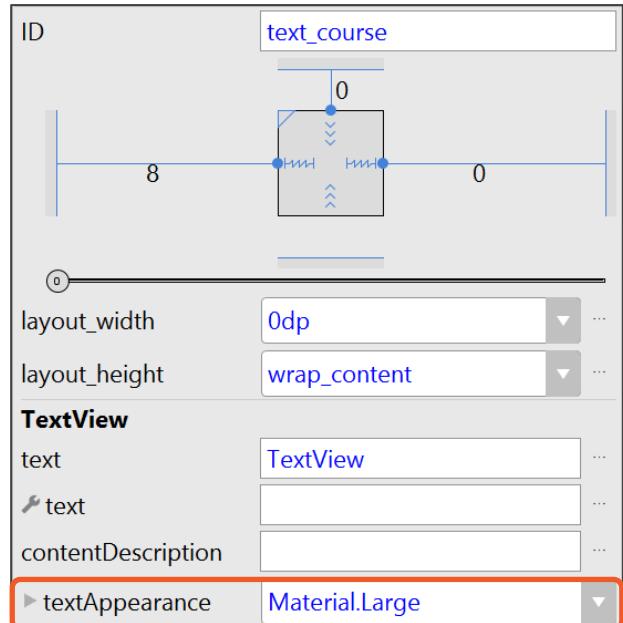
- Properties identify appearance attributes
- Often set on each view individually

## **Styles define a collection of attributes**

- Define and name attribute collection
- Associated with specific views
  - View takes on attributes in collection



# View Styles



# Declaring View Styles

## Styles declared as a values resource

- Declared with style element
- Child elements provide style attributes
  - Each declared with item element
  - Identified with name



# Declaring View Styles

```
<resources>  
    <style>          >  
        <item>          >      </item>  
        <item>          >      </item>  
    </style>  
</resources>
```



# Declaring View Styles

## Each view supports specific style attributes

- Provided in view class' documentation
- See the “XML attributes” section

### TextView class' XML attributes

XML attributes	
<code>android:autoLink</code>	Controls whether links such as urls and email addresses are automatically found and converted to clickable links.
<code>android:autoSizeMaxTextSize</code>	The maximum text size constraint to be used when auto-sizing text.
<code>android:autoSizeMinTextSize</code>	The minimum text size constraint to be used when auto-sizing text.
<code>android:autoSizePresetSizes</code>	Resource array of dimensions to be used in conjunction with <code>autoSizeTextType</code> set to <code>uniform</code> .
<code>android:autoSizeStepGranularity</code>	Specify the auto-size step size if <code>autoSizeTextType</code> is set to <code>uniform</code> .
<code>android:autoSizeTextType</code>	Specify the type of auto-size.
<code>android:autoText</code>	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
<code>android:breakStrategy</code>	Break strategy (control over paragraph layout).
<code>android:bufferType</code>	Determines the minimum type that <code>getText()</code> will return.
<code>android:capitalize</code>	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.
<code>android:cursorVisible</code>	Makes the cursor visible (the default) or invisible.
<code>android:digits</code>	If set, specifies that this TextView has a numeric input method and that these specific characters are the ones that it will accept.
<code>android:drawableBottom</code>	The drawable to be drawn below the text.
<code>android:drawableEnd</code>	The drawable to be drawn to the end of the text.



# Declaring View Styles

**Style declaration can inherit another style**

- Can inherit framework or project styles
- Include parent on style element
- New style will have parent attributes
- Can add/override attributes



# Declaring View Styles

```
<resources>  
    <style name="myBrandedTextStyle" >  
        <item name="android:textColor">#f05a28</item>  
        <item name="android:textAllCaps">true</item>  
    </style>  
</resources>
```



# Declaring View Styles

**Shortcut syntax for inheriting a project style**

- Can use style naming to indicate parent



# Declaring View Styles

```
<resources>  
    <style name="myBrandedTextStyle" parent="myTextStyle">  
        <item name="android:textColor">#f05a28</item>  
        <item name="android:textAllCaps">true</item>  
    </style>  
</resources>
```



# Declaring View Styles

```
<resources>  
    <style name=  
        <item name="android:textColor">#f05a28</item>  
        <item name="android:textAllCaps">true</item>  
    </style>  
</resources>
```



# Applying a Style to A View

## **Set view's style property to style name**

- Applies supported attributes
- Ignores any unsupported attributes

## **Some view's have additional properties**

- Apply a specific subset of attributes
- Example:
  - TextView's textAppearance property

## **Style only applies to the specific view**

- Does not affect child/descendant views



# Themes

## Broadly applied styles

- Can be applied at the activity level
- Can be applied at application level

## Applying theme to activity

- Affects the activity
- Affects the views within the activity

## Applying theme to application

- Defines default theme for app activities



# Themes

## Applying a theme

- Set in application manifest
- Setting a specific activity's theme
  - Use theme attribute activity
- Setting the default theme
  - Use theme attribute of application



# Themes

## Themes defined as a style resource

- Usually inherit from an existing theme

## Setting theme attributes

- Can edit style resource directly
- Can use Android Studio Theme Editor
  - Provides UI for commonly modified theme attributes
  - Provides preview of theme effects



# Navigation Drawer

**Provides app's main navigation options**

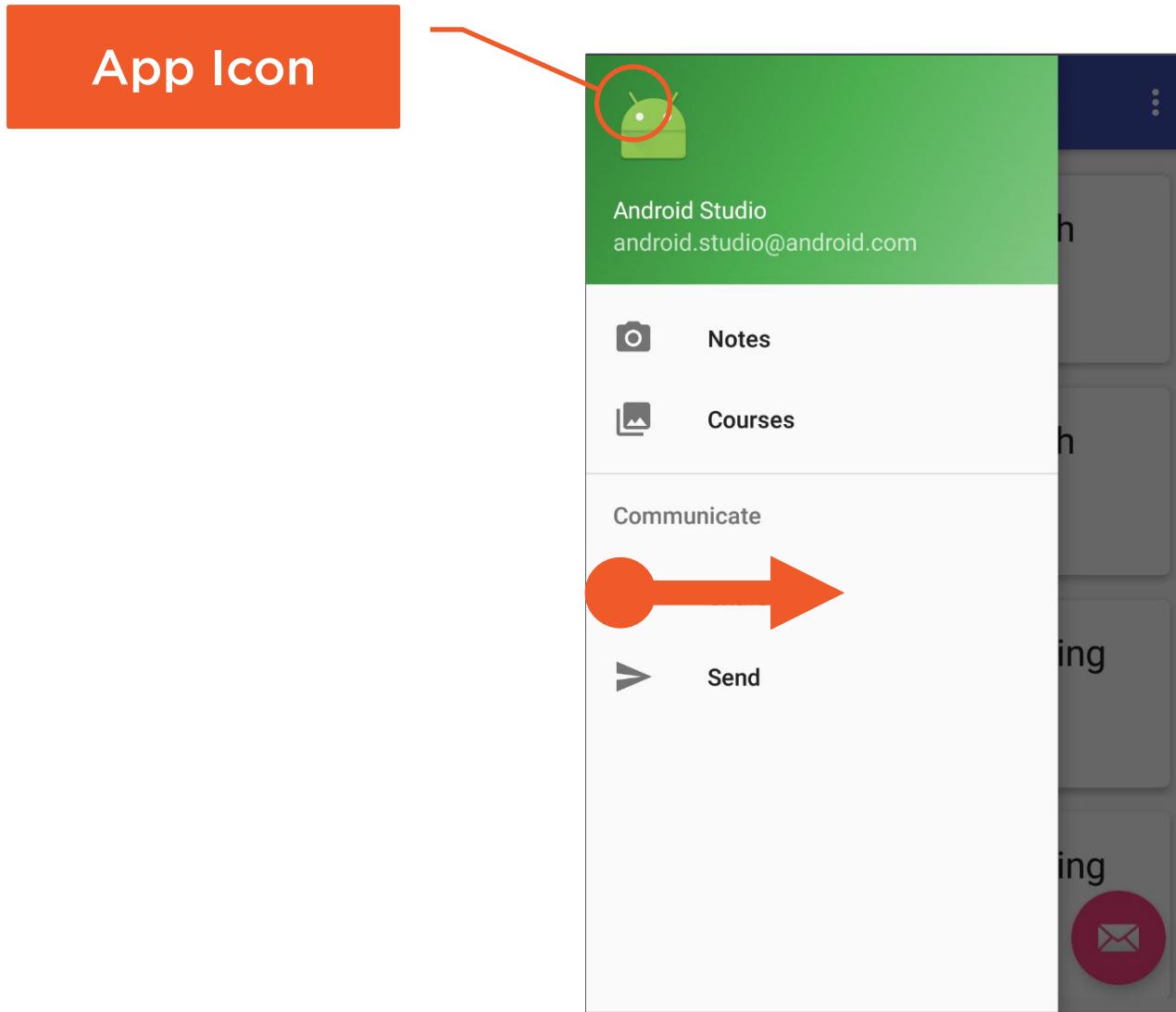
- Hidden when not in use
- Normally slides from screen's "start" edge
  - Left edge for most devices

**Accessing navigation drawer**

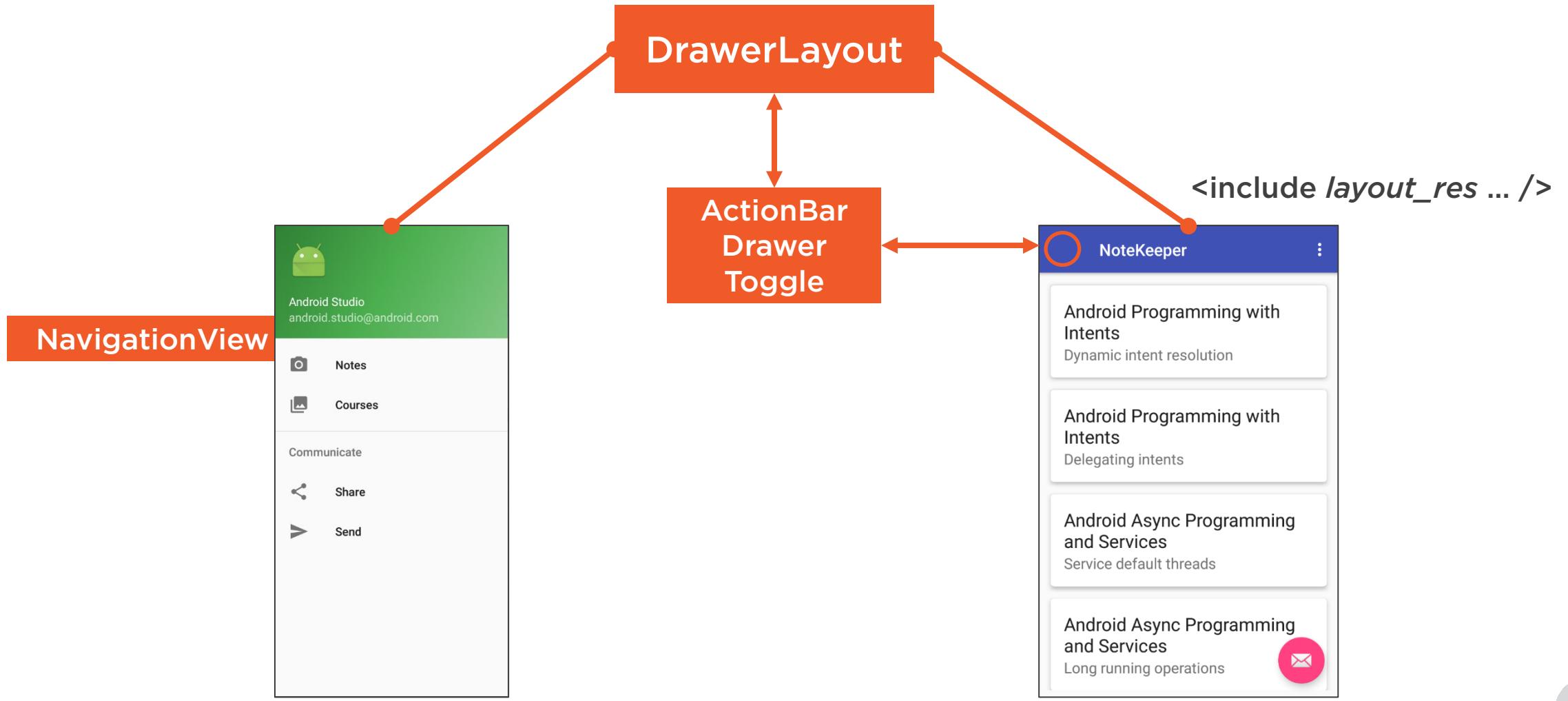
- User can open with app icon
- User can swipe left edge in/out



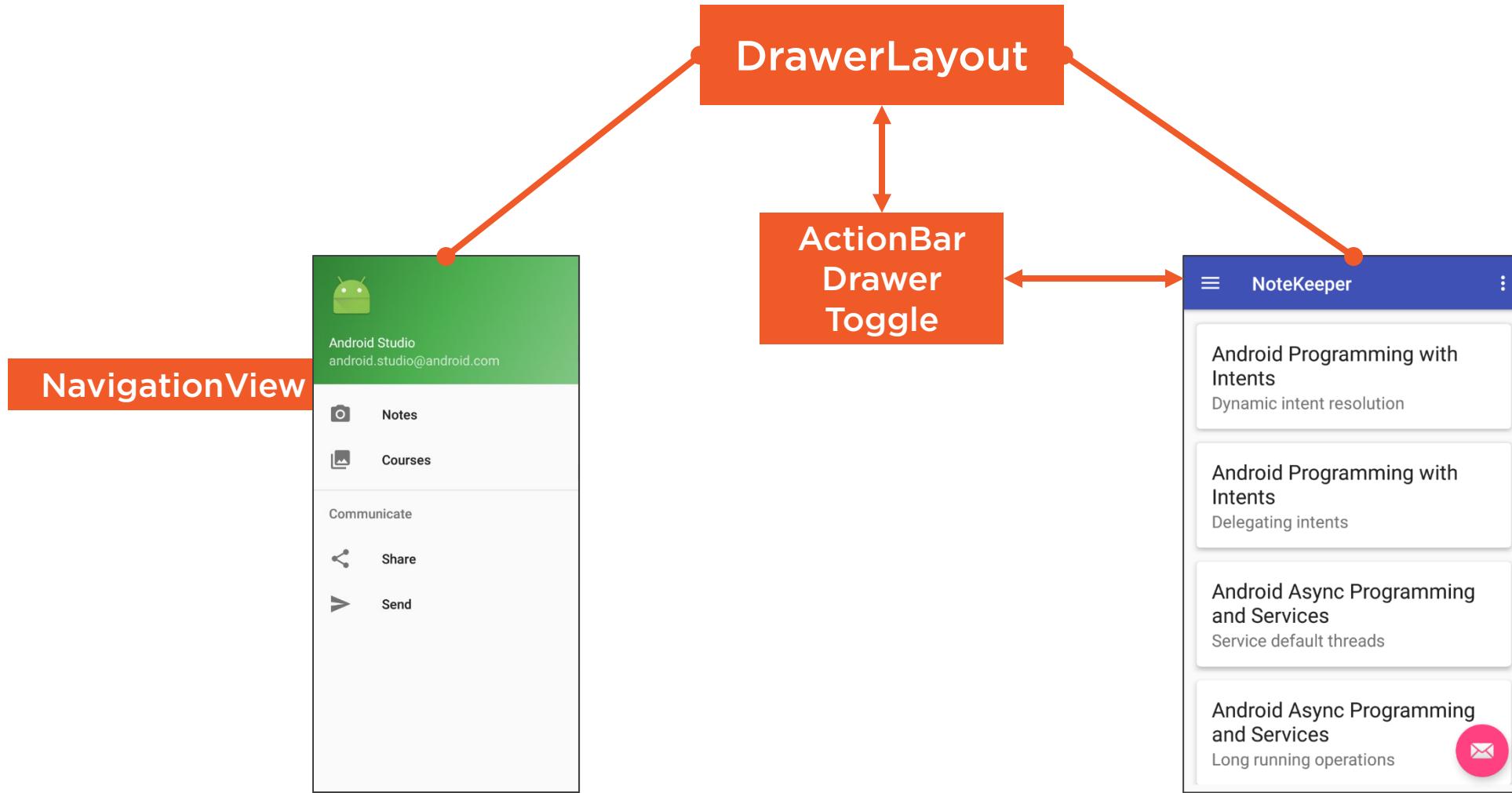
# Navigation Drawer



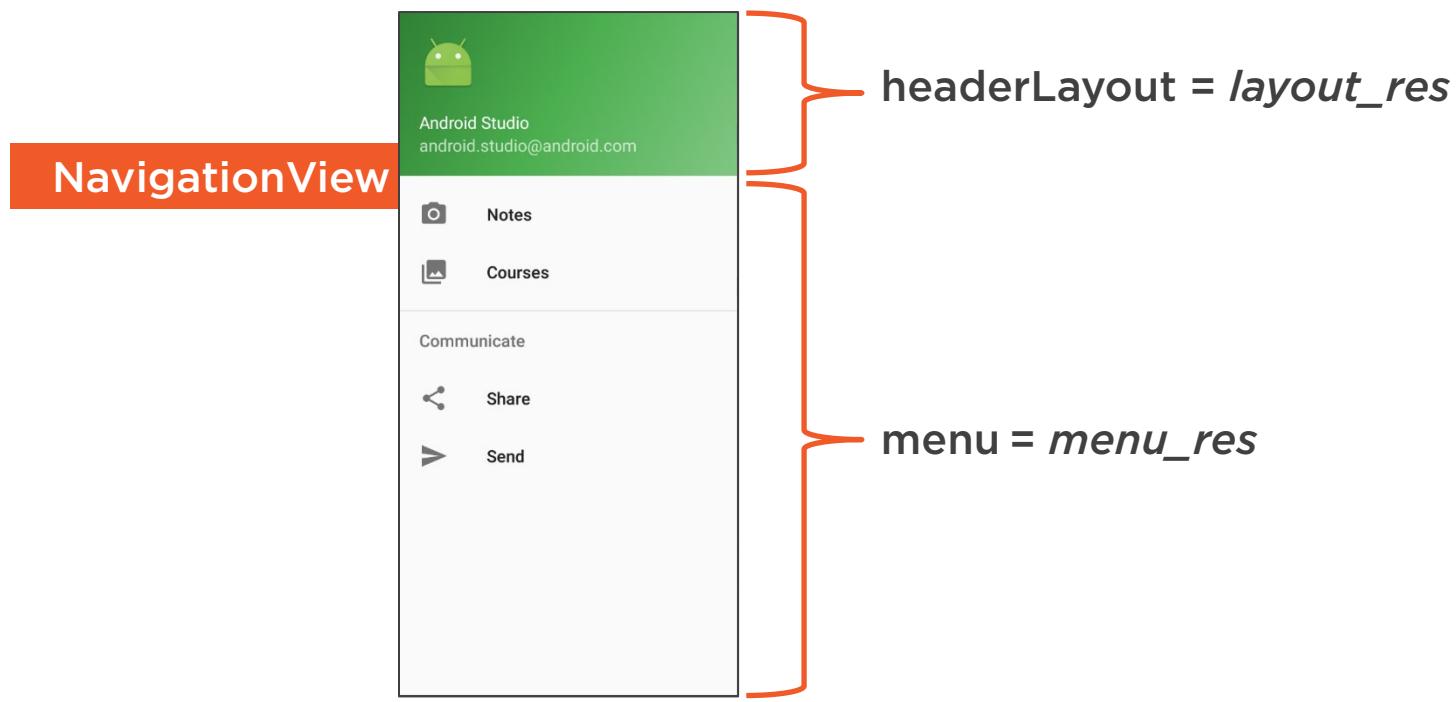
# Navigation Drawer



# Navigation Drawer NavigationView



# Navigation Drawer NavigationView



# Handling NavigationView Selections

## Implement NavigationView listener interface

- onNavigationItemSelectedListener
- One method onNavigationItemSelected
  - Receives MenuItem reference

## Associate with NavigationView

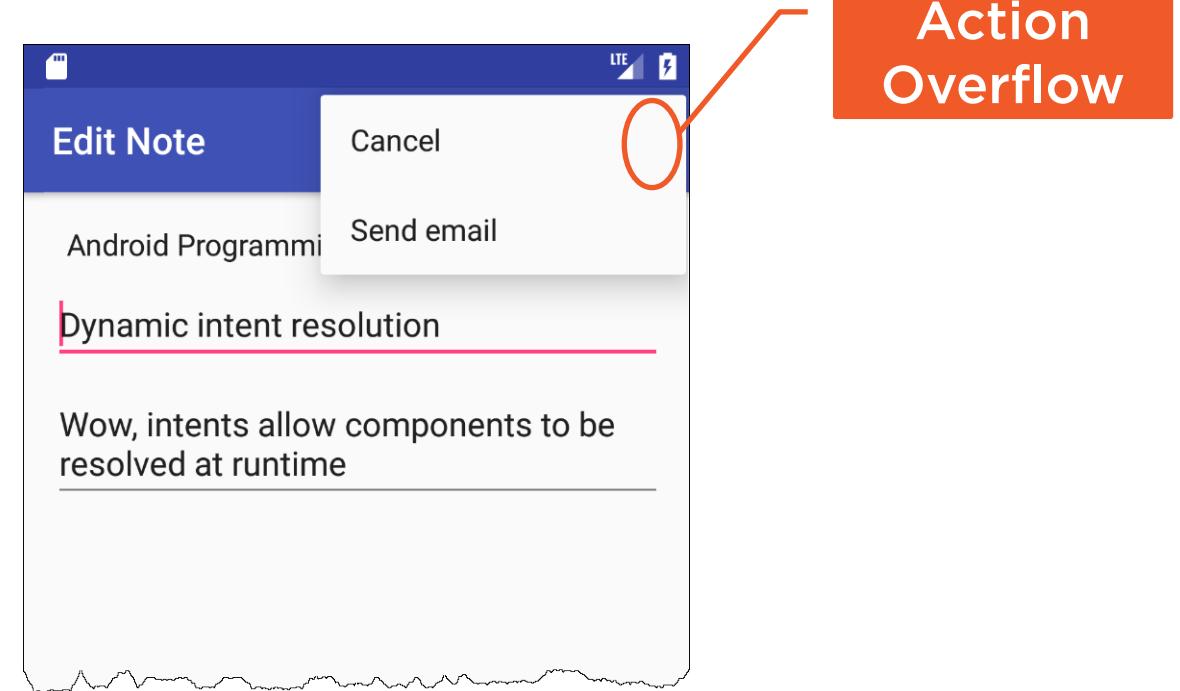
- setNavigationItemSelectedListener



# Options Menus

## Provide actions for an Activity

- Actions available in app bar
- Appear under action overflow by default



# Options Menus

## **Defined in menu resource**

- Root of component tree is menu

## **Each action defined as a menu item**

- Has unique ID within menu
- Has a text title



# Associating an Options Menu with an Activity

## **onCreateOptionsMenu**

- Receives a menu reference
- Attach menu items to menu
- Inflate menu resource with menu inflator
  - Access with getMenuInflater method



# Handling Options Menu Item Selections

## **onOptionsItemSelected**

- Receives MenuItem reference
- Retrieve MenuItem ID value
  - Access with MenuItem.getItemId
- Perform work based on ID value



# Menu Items as App Bar Actions

## Action overflow menu item challenges

- Not immediately discoverable
- Access takes multiple steps

## App bar actions

- Menu items visible on app bar
- Improve access to common menu items
- Normally have icon associated

## Making menu item an app bar action

- Use showAsAction property



# Common ShowAsAction Values

## **ifRoom**

- Display as action when space allows
- Given preference in top-to-bottom order

## **always**

- Always display as action
- Use very sparingly

## **withText**

- Show text with action when space allows
- Can be combined with ifRoom or always



# Changing Menu Items at Runtime

**Application state can change menu options**

- May need to add/remove menu items
- May need to enable/disable menu items



# Changing Menu Items at Runtime

## **onPrepareOptionsMenu**

- Override to modify menu state
- Receives reference to current menu
- Initially called before menu displayed

## **invalidateOptionsMenu**

- Call when menu state may need to change
- Schedules call to onPrepareOptionsMenu



# RecyclerView Adapter

## Create item views

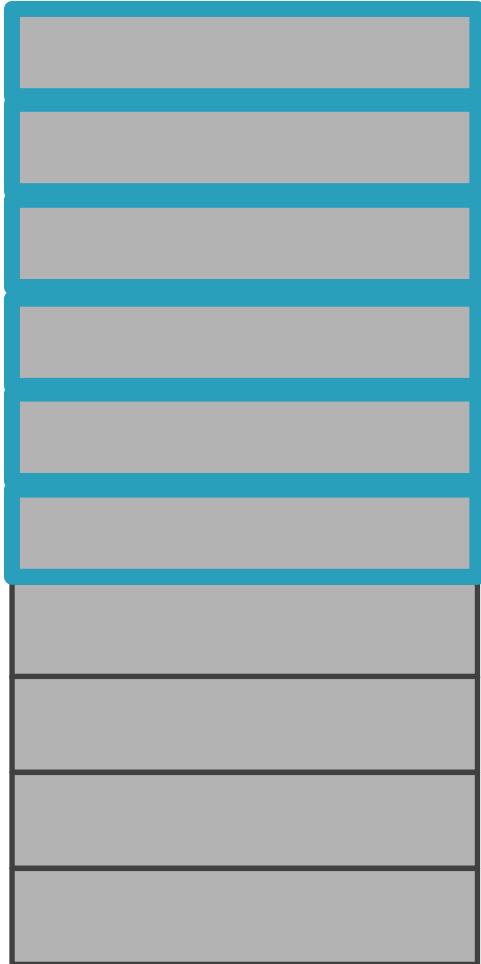
- Returned to RecyclerView
- RecyclerView manages as a pool

## Populate item views

- Received from RecyclerView pool



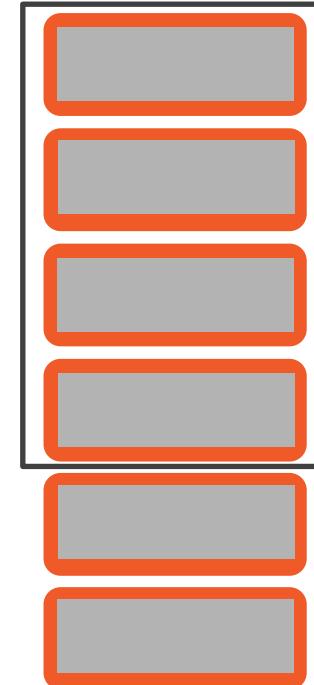
Data



# RecyclerView Adapter

Adapter

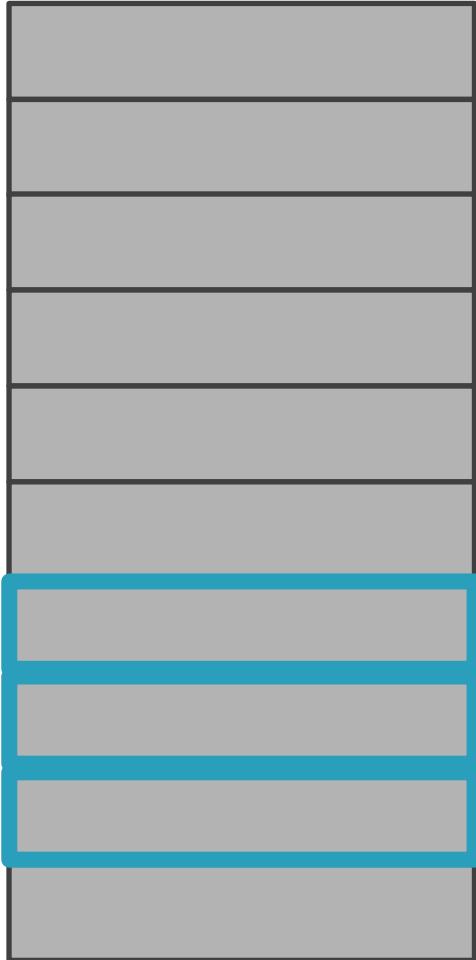
View



RecyclerView

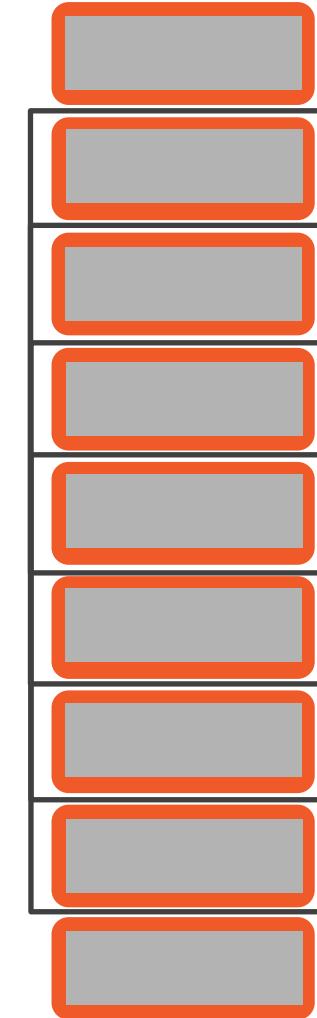
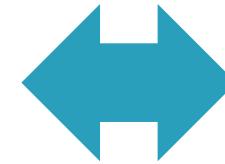
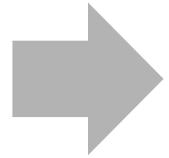


Data



# RecyclerView Adapter

Adapter



RecyclerView



# Item View Holders

## **Views managed using view holder pattern**

- Holds a reference to top-level view
- Holds references to contained views

## **Use a custom view holder class**

- Extends RecyclerView.ViewHolder
- Fields for contained views



# Implementing RecyclerView Adapter

## **Extend RecyclerView.Adapter**

- Pass view holder class as type parameter
- View holder class normally nested within



# Implementing RecyclerView Adapter

## **getItemCount**

- Return number of items

## **onCreateViewHolder**

- Create item view
- Store item view references in view holder

## **onBindViewHolder**

- Receives view holder and display position
- Set display values using view holder



# RecyclerView Item Selection

## **ListView item selection**

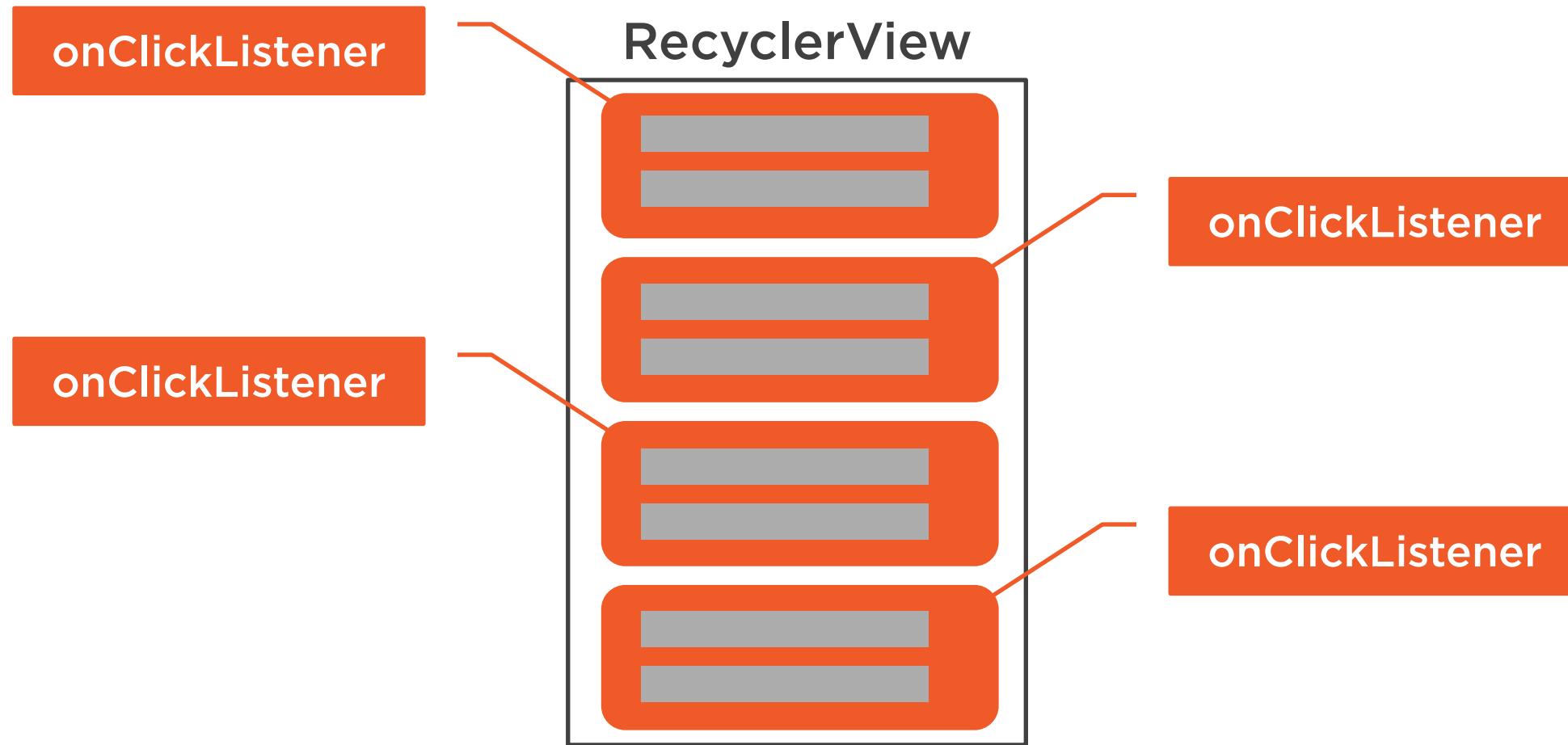
- Used AdapterView.OnItemClickListener
- RecyclerView takes different approach

## **RecyclerView has no explicit support**

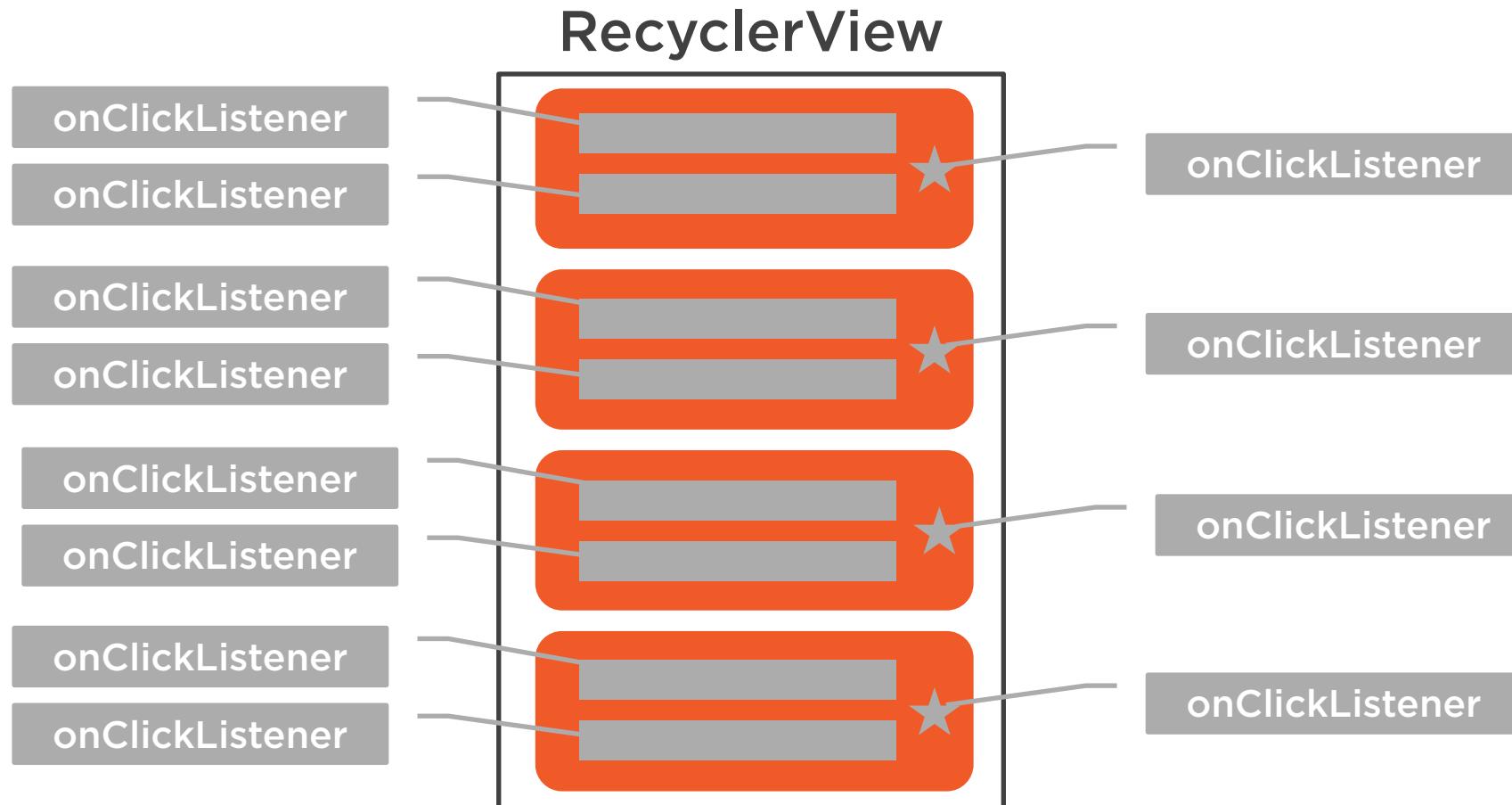
- Contained views handle click event
- Allows single item selection
- Allows multiple selections within items



# RecyclerView Item Selection



# RecyclerView Item Selection

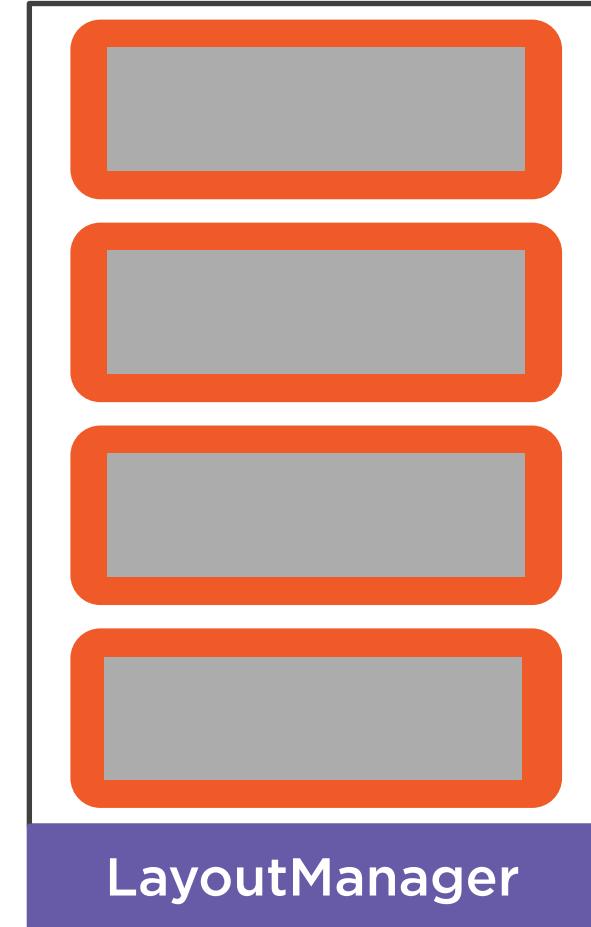


# Data



# RecyclerView

## RecyclerView



## Adapter

## View



# Android Testing

## **Android applications**

- Java-based behavior
- Android-based behavior

## **Testing Java-based behavior**

- Local JVM tests

## **Testing Android-based behavior**

- Instrumented tests



# Instrumented Tests

## Run on an emulator or physical device

- Have the full Android environment

## Instrumented Unit Tests

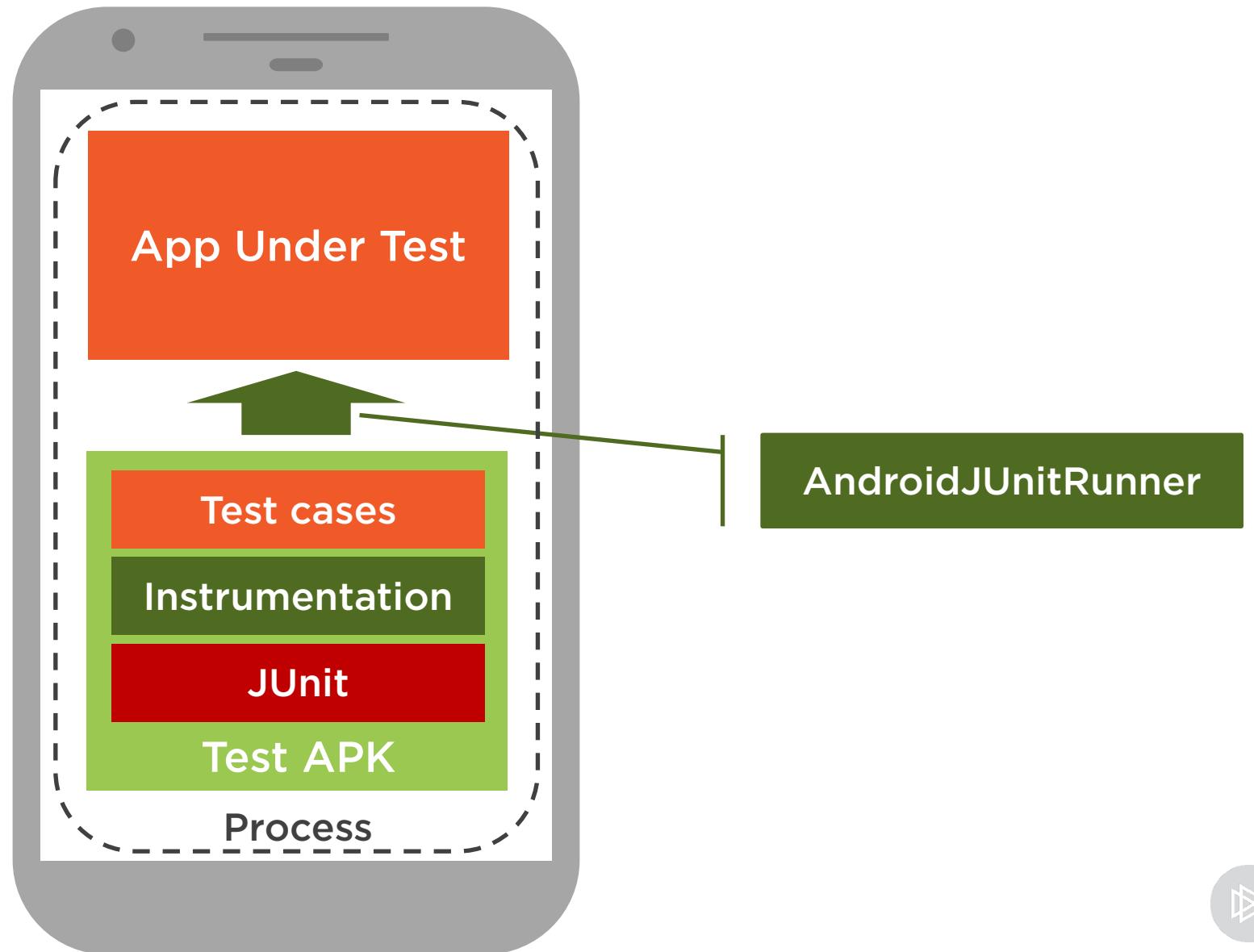
- Unit tests
- Rely on Android features/capabilities

## Automated User Interface Tests

- Integration tests
- App behaviors in response to UI actions



# Instrumented Tests



# Implementing Instrumented Tests

## Uses JUnit 4

- Test methods marked with @Test
- Supports pre/post-processing methods

## Uses Assert class

- Indicate expectations
- Fails test if expectations not met

## Test managed with Android Studio

- Can run or debug tests
  - Single test, group of tests, or all tests
- Displays tests results



# Implementing Instrumented Tests

**Organized separate from JVM tests**

- In androidTest source set

**Relies on Android JUnit test runner**

- Class must have @RunWith annotation
  - Pass AndroidJUnit4.class

**Requires Android environment**

- Run on emulator or device



# Implementing Instrumented Tests

```
@RunWith(AndroidJUnit4.class)
public class MyExampleTestClass {
    @BeforeClass
    public static void classSetUp() { . . . }
    @Before
    public void testSetUp() { . . . }
    @Test
    public void myTestMethod() {
        // Android dependent test code
    }
}
```



# Creating UI Test Interactions

## UI tests require a series of view interactions

- Need way to specify view of interest
- Need way to specify action on the view

## Espresso.onView method

- Accepts a Matcher parameter
  - Specifies view matching criteria
- Returns a ViewInteraction reference
  - Associated with matching view
  - Used to perform action on view



# Specifying View of Interest

## Uses Hamcrest matchers

- Provides declarative matching
- General purpose Java framework
- <http://hamcrest.org>

## ViewMatchers class

- Provides matchers for Android Views
- Methods return a Hamcrest matcher
- Easily combined with Hamcrest general purpose matchers



# Specifying View of Interest

## Example ViewMatchers methods

- (withId
  - Match views based on id property
- (withText
  - Match views based on text property
- (isDisplayed
  - Match views currently on screen
- (isChecked
  - Match currently checked checkable views (Switch, CheckBox, etc.)



# Specifying View of Interest

## Example Hamcrest Matchers

- equalTo
  - Match based on equals method
- instanceOf
  - Match based on class type
- allOf
  - Accepts multiple Matchers
  - Match if all Matchers match
- anyOf
  - Accepts multiple Matchers
  - Match if any Matchers match



# Performing View Action

## **ViewInteraction.perform method**

- Performs one or more specified actions
- Specific action passed as a parameter

## **ViewActions class**

- Provides action methods
- Each method returns specified action



# Performing View Action

## Example ViewActions methods

- click
  - Click on the view
- typeText
  - Type text into view
- replaceText
  - Replace view's text
- closeSoftKeyboard
  - Closes the soft keyboard



# Starting the Target Activity

## ActivityTestRule

- Automates test activity lifetime
- Starts activity before each test
- Terminates activity after each test
- Activity life includes @Before/@After methods

## Using ActivityTestRule

- Declare & initialize as test class field
- Desired activity as type parameter
- Mark field with @Rule annotation



# Testing Views That Use Adapters

## **AdapterView derived views**

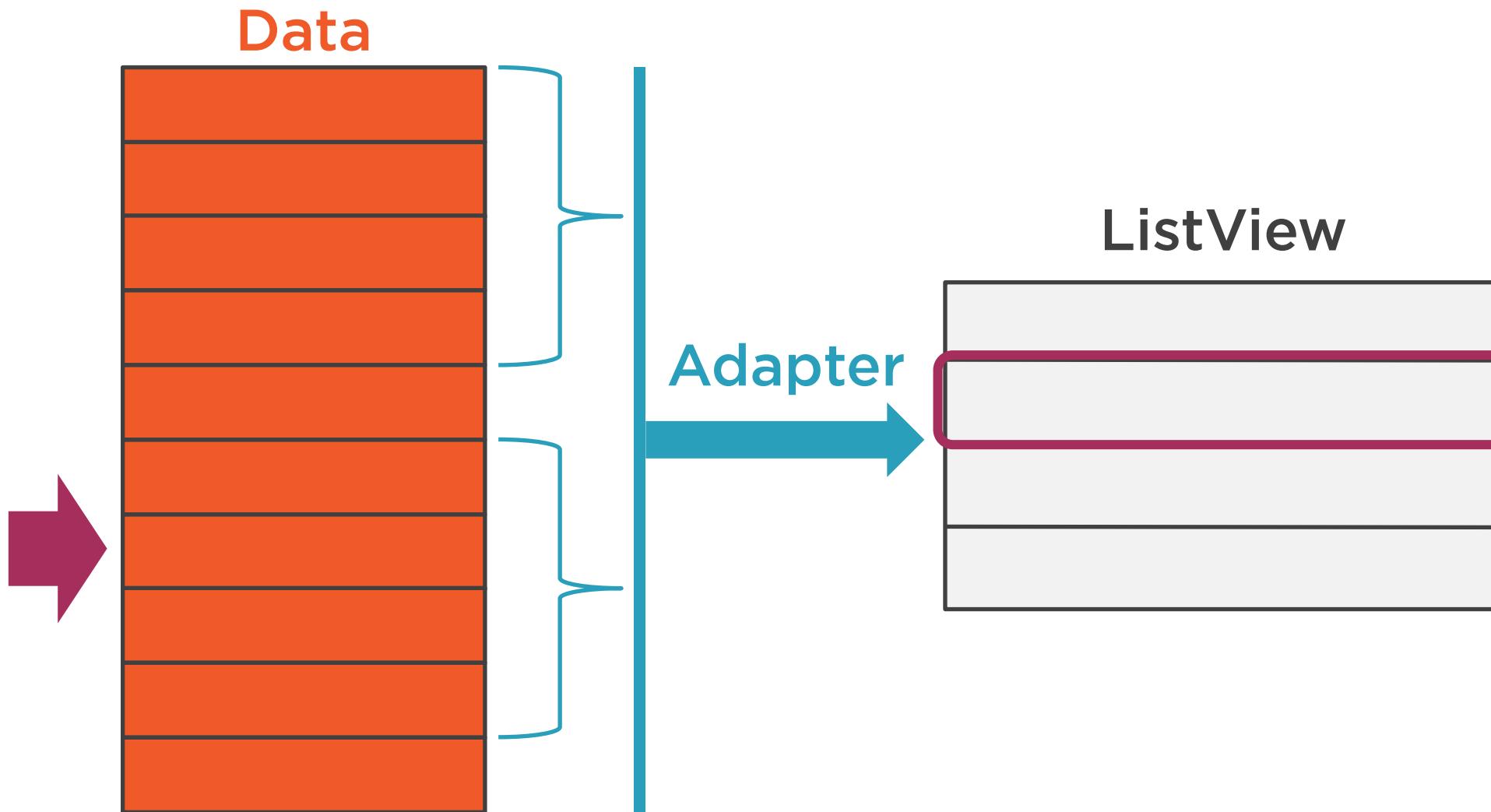
- Load data from Adapter classes
- Examples include ListView & Spinner

## **These views display multiple data items**

- Only a subset may be loaded
- Test selection based on target data



# Testing Views that Use Adapters



# Testing Views That Use Adapters

## `Espresso.onData`

- Specify matcher based on target data
- Tend to use general purpose matchers

## `DataInteraction`

- Provides methods for interacting with or narrowing match

## Tend to use `DataInteraction.perform`

- Performs action on top-level view for entry in AdapterView



# Back Button

## **Espresso.pushBack**

- Performs action of pushing back button
- No reference to a view needed



# Verifying Behavior

**Tests meant to confirm expected behavior**

- UI Behavior
- Logic Behavior



# Verifying UI Behavior

## **ViewInteraction.check method**

- Confirms some aspect of a view

## **ViewAssertions class**

- Provides view assertion methods



# Verifying UI Behavior

## Common ViewAssertions methods

- matches
  - Confirms view matches passed matcher
  - Commonly used with ViewMatchers
  - Also confirms that view exists
- doesNotExist
  - Confirms that view does not exist



# Verifying Logic Behavior

**Assert methods still important**

- Main way we confirm logic behavior



# Verifying Logic Behavior

**Remember that UI tests do two jobs**

- Run testing behavior
- Run app behavior
- Test behavior interacts with app

**Sequencing is important**

- Each test action must wait for corresponding app action to complete
- Handled by ViewInteractions methods
  - i.e. perform, check, etc.
- Assert methods do not



# Verifying Logic Behavior

## Instrumentation class

- Allows interaction with test environment
- Access with InstrumentationRegistry
  - Use getInstrumentation method

## Instrumentation.waitForIdle

- Accepts Runnable reference
- Provides code coordination
- Waits for app action triggered by previous test action to complete



# Verifying UI Behavior

## **ViewInteraction.check method**

- Checks that the view conforms to some assertion
- Accepts a ViewAssertion reference

## **ViewAssertions class**

- Provides methods for asserting some aspect of a view
- Methods create a ViewAssertion



# Specifying View of Interest

## Example ViewMatchers methods

- (withId
  - Match view based on id property
- (withText
  - Match view based on text property
- (isDisplayed
  - Match views currently on screen
- (isChecked
  - Match currently checked checkable views (Switch, CheckBox, etc.)



# Testing

## **Testing needs to be a core task**

- Essential to delivering quality software

## **Functional testing**

- Verify behaves as expected
- Detect breaking changes



# Testing

## Unit testing

- Testing of units of code
- Each unit test is relatively simple
  - Tests a specific feature/behavior
- Generally will have many unit tests

## Integration testing

- Testing of pieces being put together
- Application behaviors
- Often involves testing of UI



# Unit Testing

**Unit tests should be run often**

- After code changes
- Before check-in to source code control

**Generally want to run all unit tests**

- No change is complete until all tests pass

**Ideally can be run reasonably quickly**



# Android Application Testing

## Challenges of testing Android apps

- Full testing needs Android environment
- Requires emulator or a physical device

## Need way to efficiently run unit tests

- Limit how often full environment needed



# Efficiently Running Unit Tests

## Android applications

- Java-based behavior
- Android-based behavior

## Separate tests

- Tests for Java-based behavior
- Tests for Android-based behavior

## Efficiently running unit tests

- Tests Java parts of app locally
- Leverage JVM on desktop



# Local JVM Tests

## Android Studio JVM testing

- Separate source set for JVM tests
- Uses JUnit 4

## Managing tests with Android Studio

- Can run or debug tests
  - Single test, group of tests, or all tests
- Displays tests results
  - Success/failure indicated by color



# Testing with JUnit

## **Each unit test is a separate method**

- Marked with @Test annotation
- JUnit handles details of running method

## **Tests grouped within classes**

- Primarily for organization convenience
- Allows execution grouping
- Allows setup/teardown grouping



# Testing with JUnit

## Assert class

- Use to indicate expected results
- Fails test when expectation not met



# Testing with JUnit

## Example Assert class methods

- `assertSame`
  - Two references to same object
- `assertEquals`
  - Two objects equal (equals method)
- `assertNull`
  - Reference is null

## Negative versions of most methods

- i.e. `assertNotSame`, etc.



# Assuring Test Consistency

## **Test reliability**

- Each test must always run consistently
- Can't depend on action of another test
- Can't be impacted by side effects of other tests

## **Test must always start from same state**

- Test order is not guaranteed
- Need way to set/reset test state



# Assuring Test Consistency

## Test pre-processing

- Method with @Before annotation
- Will run before each test in class

## Test post-processing

- Method with @After annotation
- Will run after each test in class



# Assuring Test Consistency

## Once per class pre-processing

- Method with `@BeforeClass` annotation
- Will run once before all tests in class
- Method must be static

## Once per class post-processing

- Method with `@AfterClass` annotation
- Will run once after all tests in class
- Method must be static



# Assuring Test Consistency

## Multiple pre/post-processing methods

- Multiples in a class valid
- All will run
- Sequence is not guaranteed



# Summary



## Testing needs to be a core task

- Essential to delivering quality software

## Unit Testing

- Units of relatively simple tests
- Focus is on specific feature/behavior
- Tests should be run frequently



# Summary



## Local JVM testing

- Focused on Java aspects of our app
- Run directly on desktop

## Managing tests in Android Studio

- Can run or debug tests
- Displays test success or failure



# Summary



## Test methods

- Grouped in classes
- Marked with @Test annotation

## Assert class

- Use to indicate expected results
- Fails test when expectation not met



# Summary



## Tests need to be reliable

- Need to assure consistent testing

## Can run test pre/post-processing

- Use @Before and @After
- Methods run for each @Test method

## Can run test class pre/post-processing

- Use @BeforeClass and @AfterClass
- Methods run once for test class
- Methods must be static



# Android Build Process



**Android build process is somewhat involved**

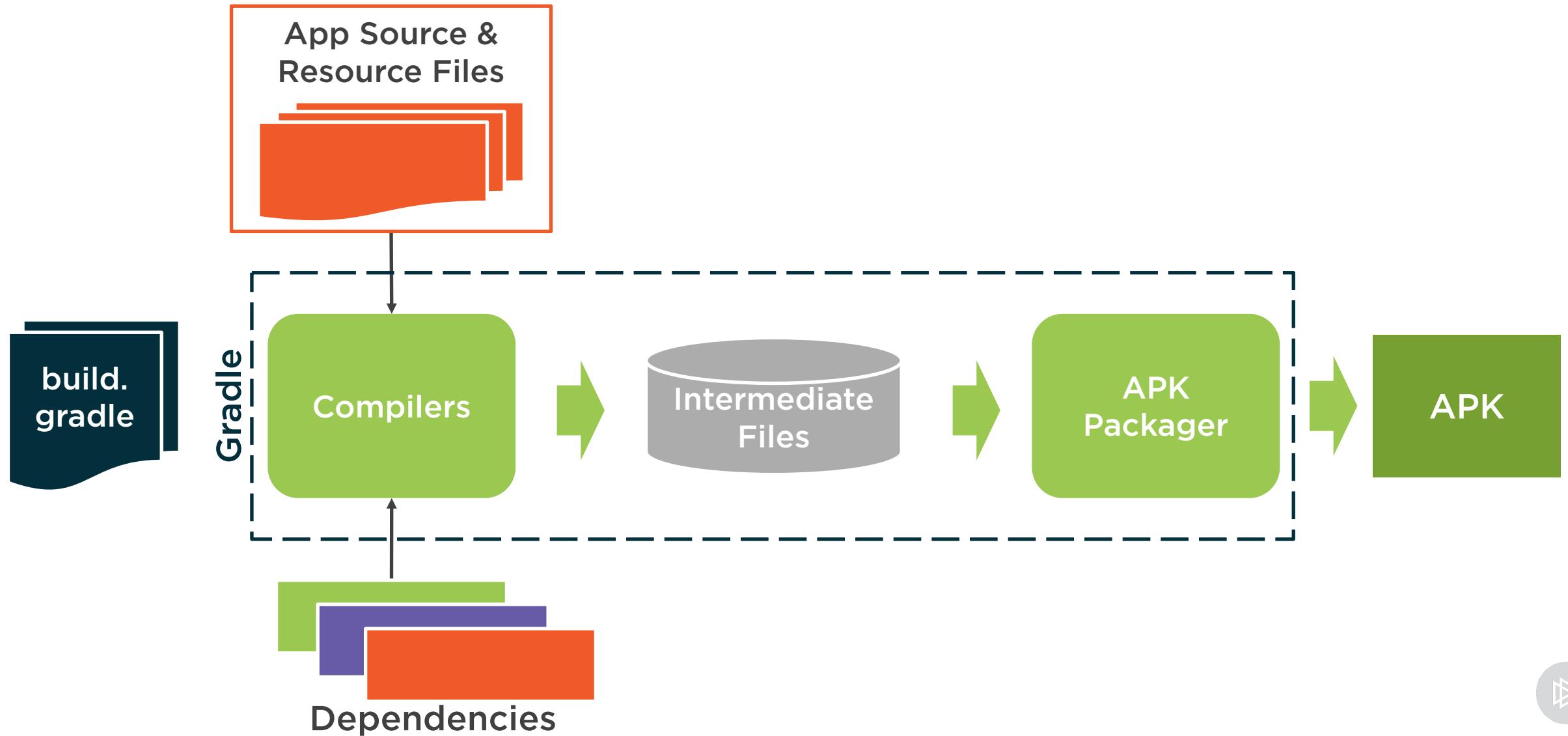
- Manually managing details is challenging

**Gradle simplifies managing build process**

- A general purpose build system
- Android-oriented features from plug-in



# Android Build Process



# Configuring Gradle

## Gradle is extremely powerful

- Very flexible
- Uses a domain specific language (DSL)

## Common settings easily managed

- Projects include build.gradle files
- Changes often simple edits

## Android Studio UI

- Many changes can be made with UI
- Use File/Project Structure...



# Dependencies

**Applications builds rarely stand alone**

- May rely on external binaries
- May rely on other project libraries

**Listing dependencies in Gradle**

- In build.gradle dependencies block
- Automatically includes dependency dependencies



# Dependency Types

## Module dependency

- Module from your project

## Jar dependency

- Java jar file

## Library dependency

- Pull from a repository



# Library Dependencies

**Will use local machine for some**

- Android Support repository
- Google repository

**Other repositories must be specified**

- Normally leverages jcenter repository
- Can add others



# Associating Dependencies

## Dependency for all build variants

- Use compile

## Dependency for JVM test

- Use testCompile

## Dependency for Instrumentation test

- Use androidTestCompile



# Android Support Library

## Backward compatibility

- Makes some newer platform features available to older platform versions
- Uses alternate classes

## Convenience and helper classes

- Provides features not part of platform
- Especially in the area of the UI

## Debugging, testing, and utilities

- Testing Support Library
- Enhanced code checks
- Special case utilities



# Android Support Library Organization

## Most grouped by platform support

- Name historically indicates min platform
- v4 Support libraries – API 4 & up
- v7 Support libraries – API 7 & up
- v13 Support libraries – API 13 & up
- Changed with latest releases
  - None support less than API 9

## Specific libraries tied to features

- Multiple libraries within each group
- We reference specific library in Gradle



# Android Support Library Organization

## Historically grouped by supported platform

- Name indicates minimum platform
- v4 Support libraries - API 4 & up
- v7 Support libraries - API 7 & up
- v13 Support libraries - API 13 & up
- Changed with latest releases
  - None support less than API 9

## Specific libraries tied to features

- Multiple libraries within each group
- We reference specific library in Gradle



# Android Build Process



**Android build process is somewhat involved**

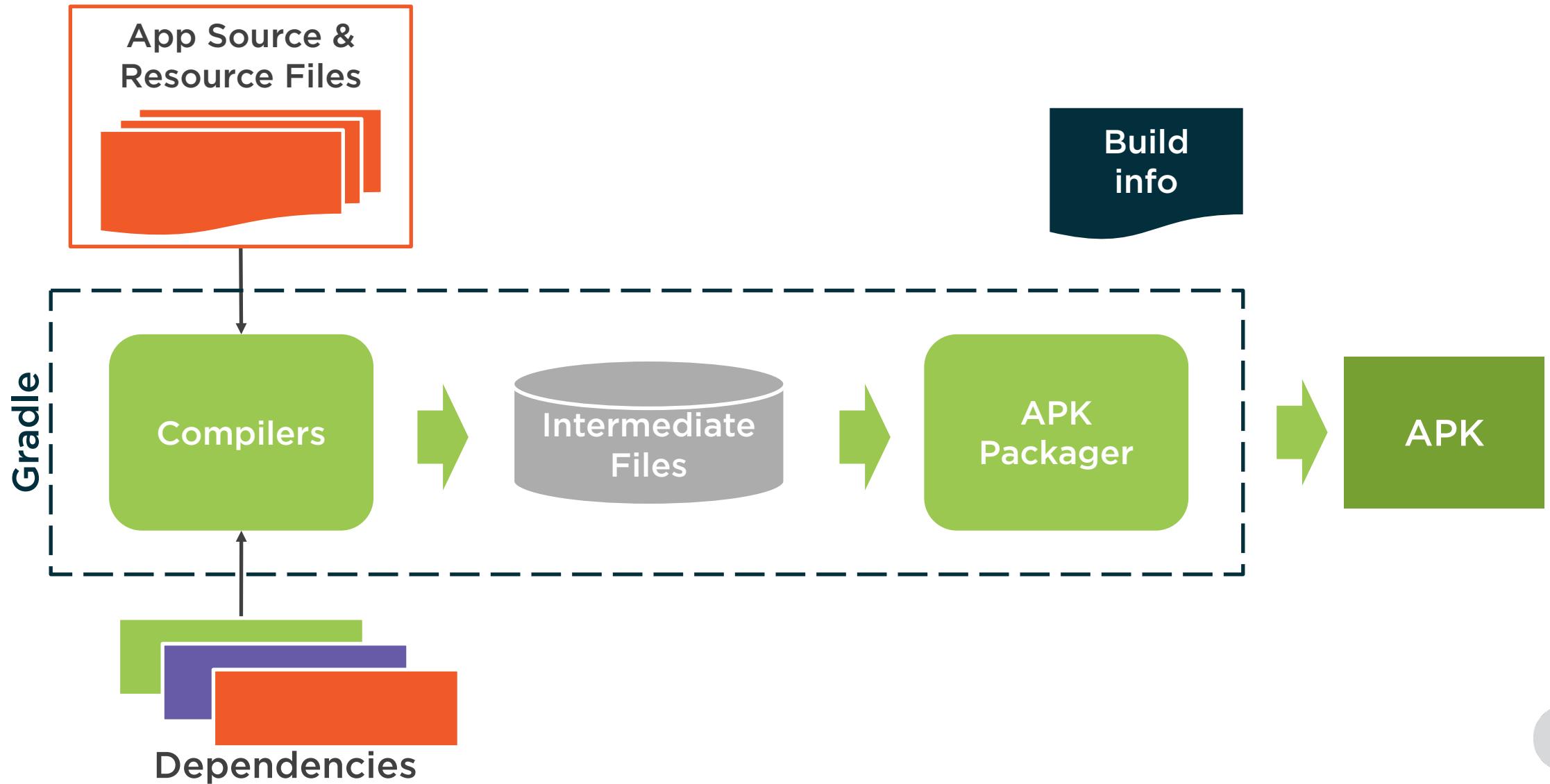
- App source/resource compilation
- App dependencies
- Producing app package (APK) file

**Gradle simplifies managing build process**

- A general purpose build system
- Android-oriented features from plug-in



# Android Build Process



# Android Studio



**Primary Android development tool**

**Handles installing the things we need**

- Android SDK
- JDK
- Other Android development tools

**Built on IntelliJ IDEA technology**



# Android Studio



**Code development & UI design**

**Developer productivity**

**Code deployment and execution**

**Debugging**

**Build system**

**Testing**





## Logcat

- System for recording log information

## Log class

- Provides methods for writing to logcat
- Includes a tag with each message
  - Commonly use class name as tag

## Android Monitor

- Displays logcat messages
- Filterable
- Searchable





## Messages are organized into levels

- Indicates relative severity

## Log class

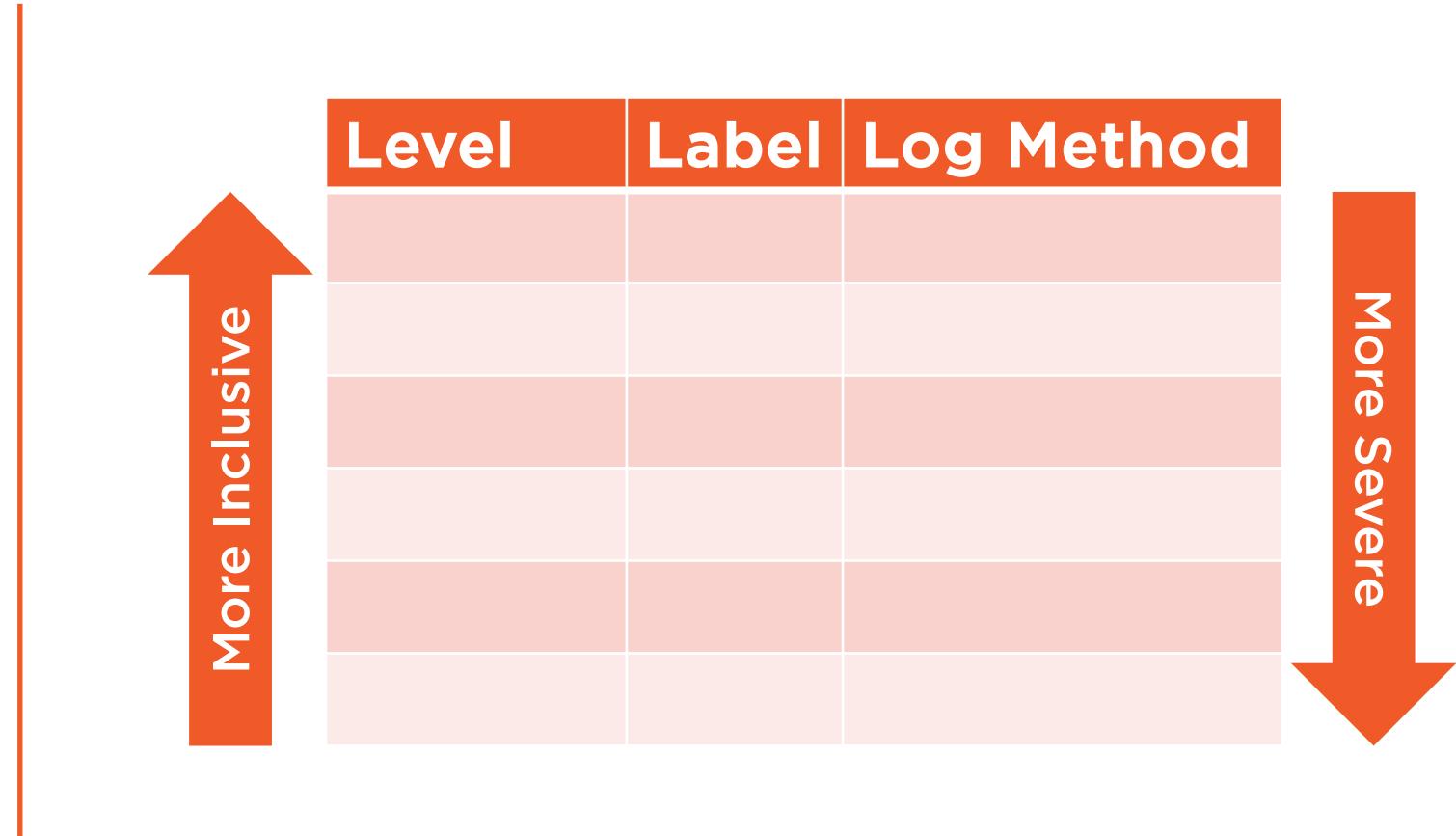
- Provides methods for each level

## Viewing messages

- Messages are labeled with level
- Can limit which levels are displayed
- Shows messages  $\geq$  selected severity



# Logcat Levels



# Summary



## Android Studio

- Central tool for Android development

## Developer Productivity

- Refactoring
- Code generation
- Customizable to developer style



# Summary



## Program execution and debugging

- Handles installing and running app
- Breakpoints and code stepping
- Viewing variables
- On-the-fly statement evaluation

## Instant Run

- Accelerates deploying code changes
- Does just enough to apply changes
- Often allows app to keep running



# Summary



## Resolving unhandled exceptions

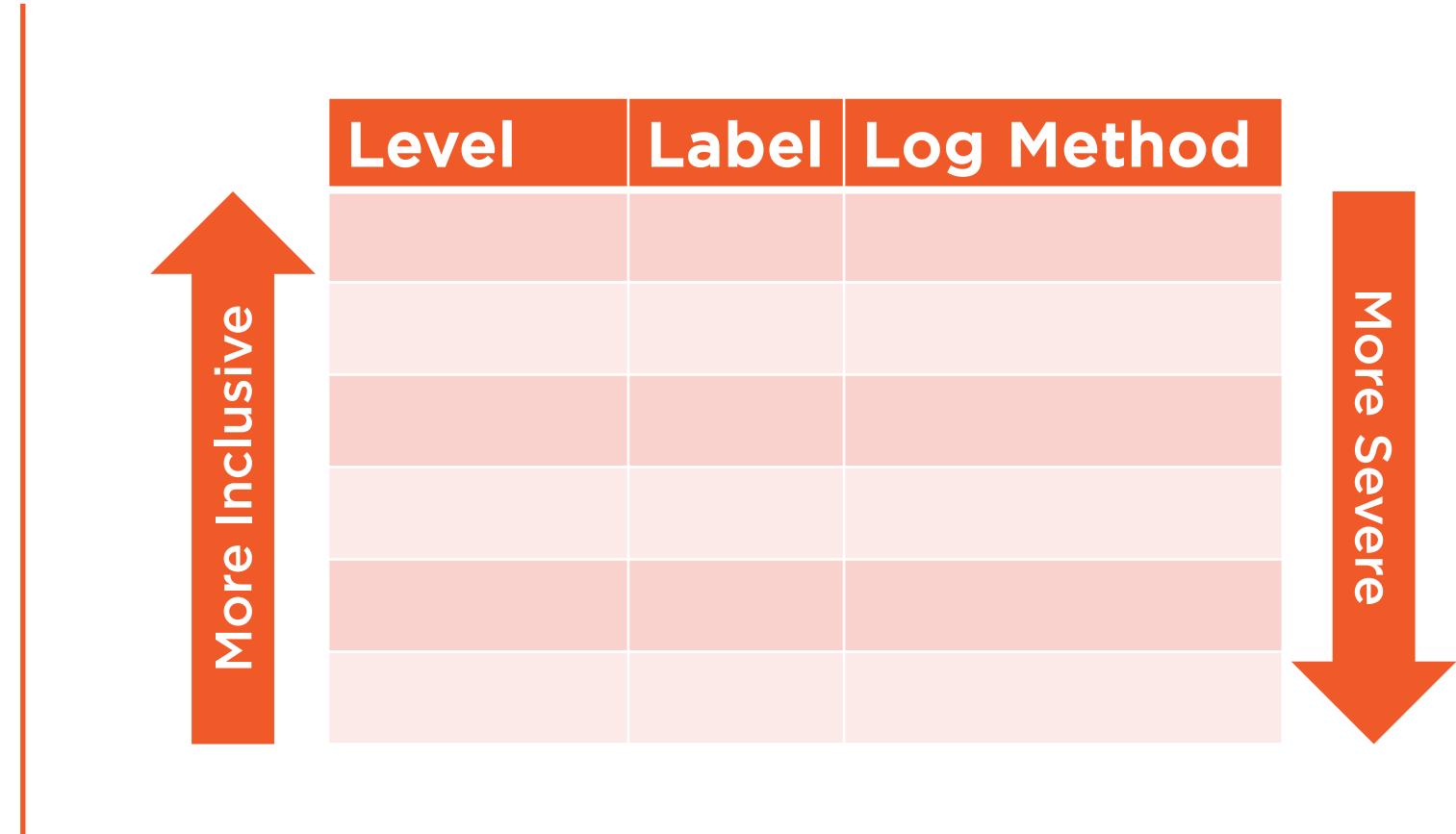
- Crash information in Android Monitor
- Shows exception and call stack
- Provides links to app source code lines

## Logcat

- System for recording log information
- Written to with Log class
- Viewable in Android Monitor



# Logcat Levels





ore



# Testing with a Real Device



# Debugging



# Application Activity Relationship

## Android is a component-oriented platform

- Components run within a process

## Process lifetime

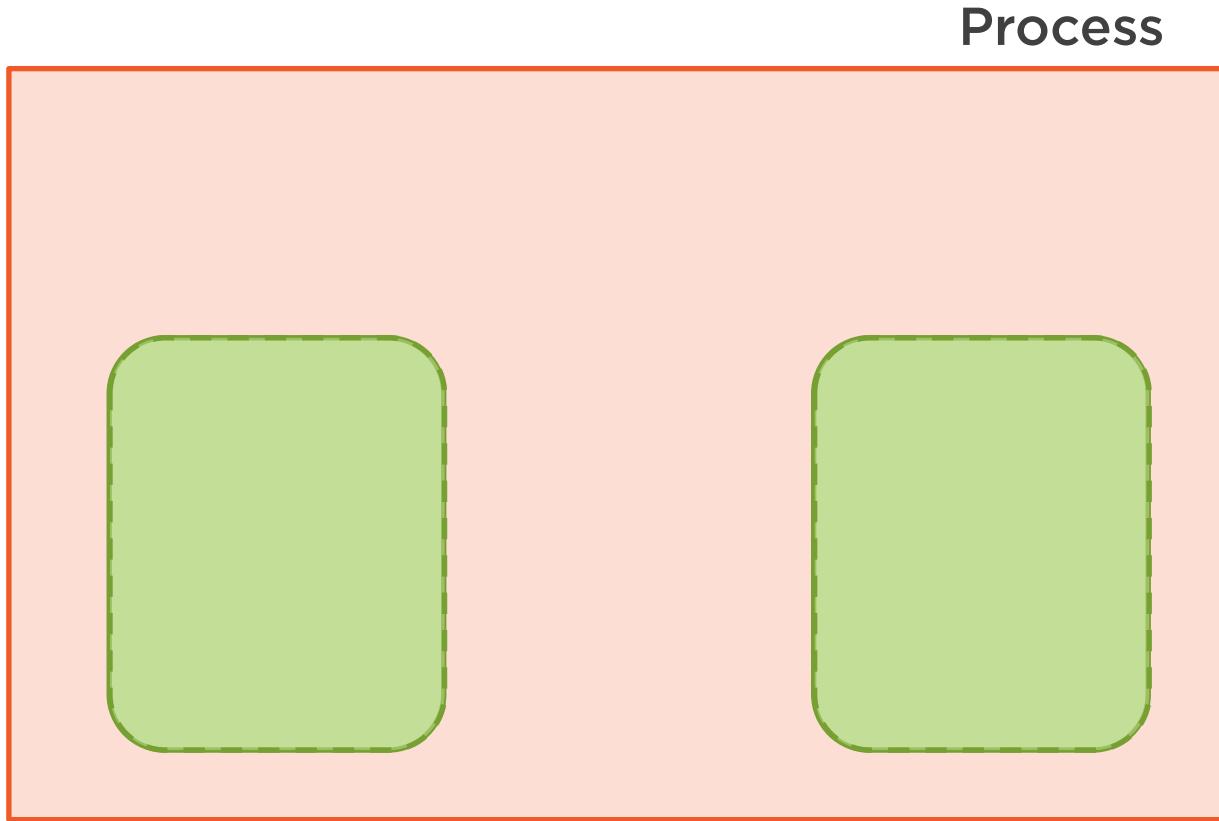
- Driven by component lifetime
- Launched for first component accessed
- Terminates after last component exits

## Application process

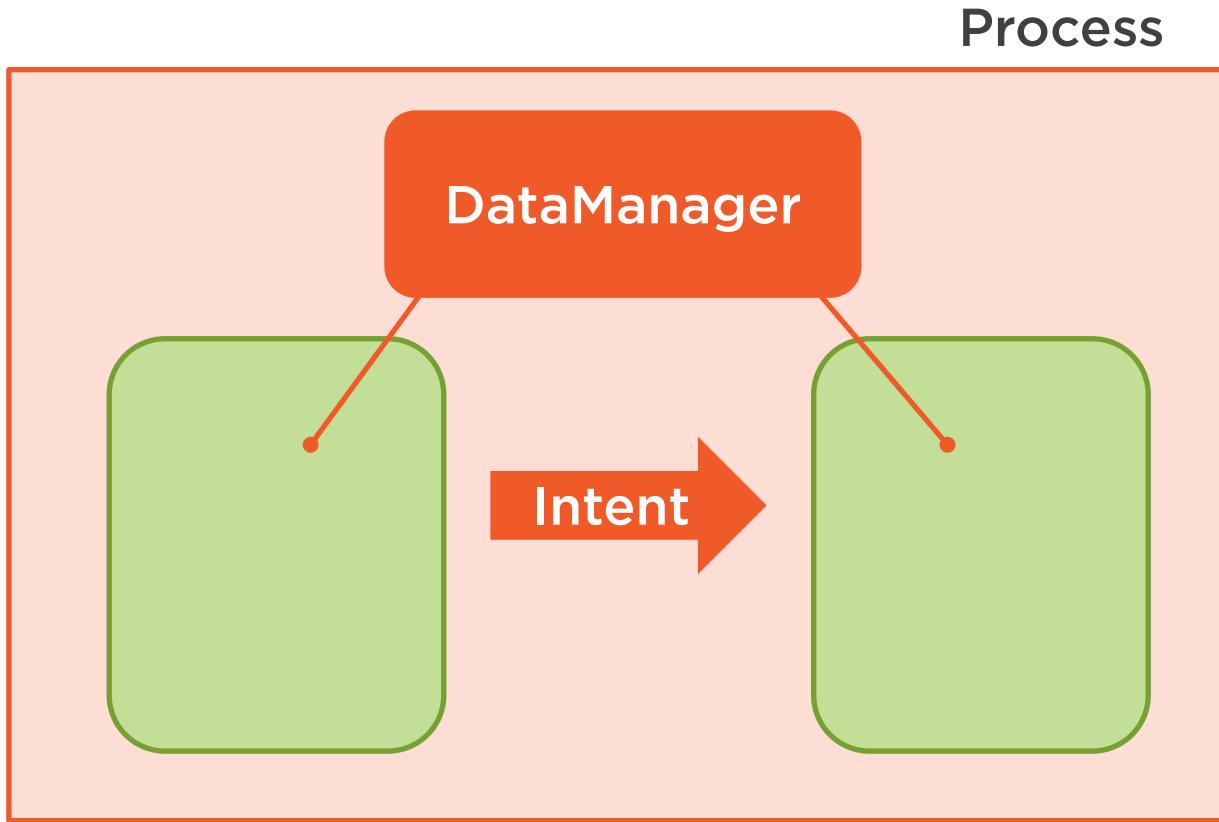
- Each app has its own process
- App components run in same process
  - When simultaneously active



# Application Activity Relationship



# Application Activity Relationship



# Late-binding Components

**Intents describe a desired operation**

- Identify operation target

**Explicit intent**

- Target is explicitly identified
- Specify the Activity class to use

**Implicit intent**

- Target is implied
- Specify the Activity characteristics



# Late-binding Components

## Implicit intents provide late-binding

- Match is determined at runtime

## System finds best match

- Often comes from another app
- Specific match may vary depending on apps installed on user device
- Prompts user if tie

## Decouples sender and receiver

- Sender may not know receiver
- Receiver may not know sender



# Implicit Intent Characteristics

## Action

- Action string
- Many standard constants available
- Example: Intent.ACTION\_VIEW
- Commonly set in Intent constructor
- Only required characteristic

## Category

- Provides extended qualification
- Not normally used by sender



# Implicit Intent Characteristics

## Data

- URI of data to be acted upon
- Example: <https://pluralsight.com>
- Set with Intent.setData

## Mime type

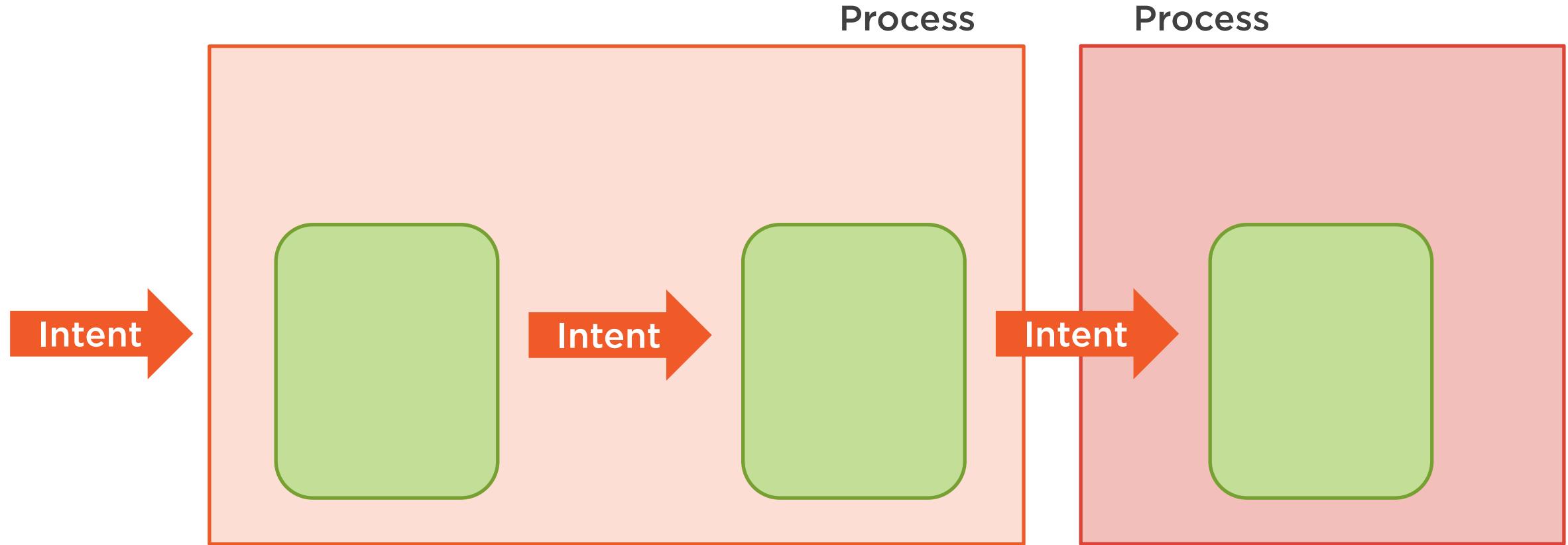
- Common or app-specific mime type
- Example: text/html
- Set with Intent.setType

## Setting data and mime type

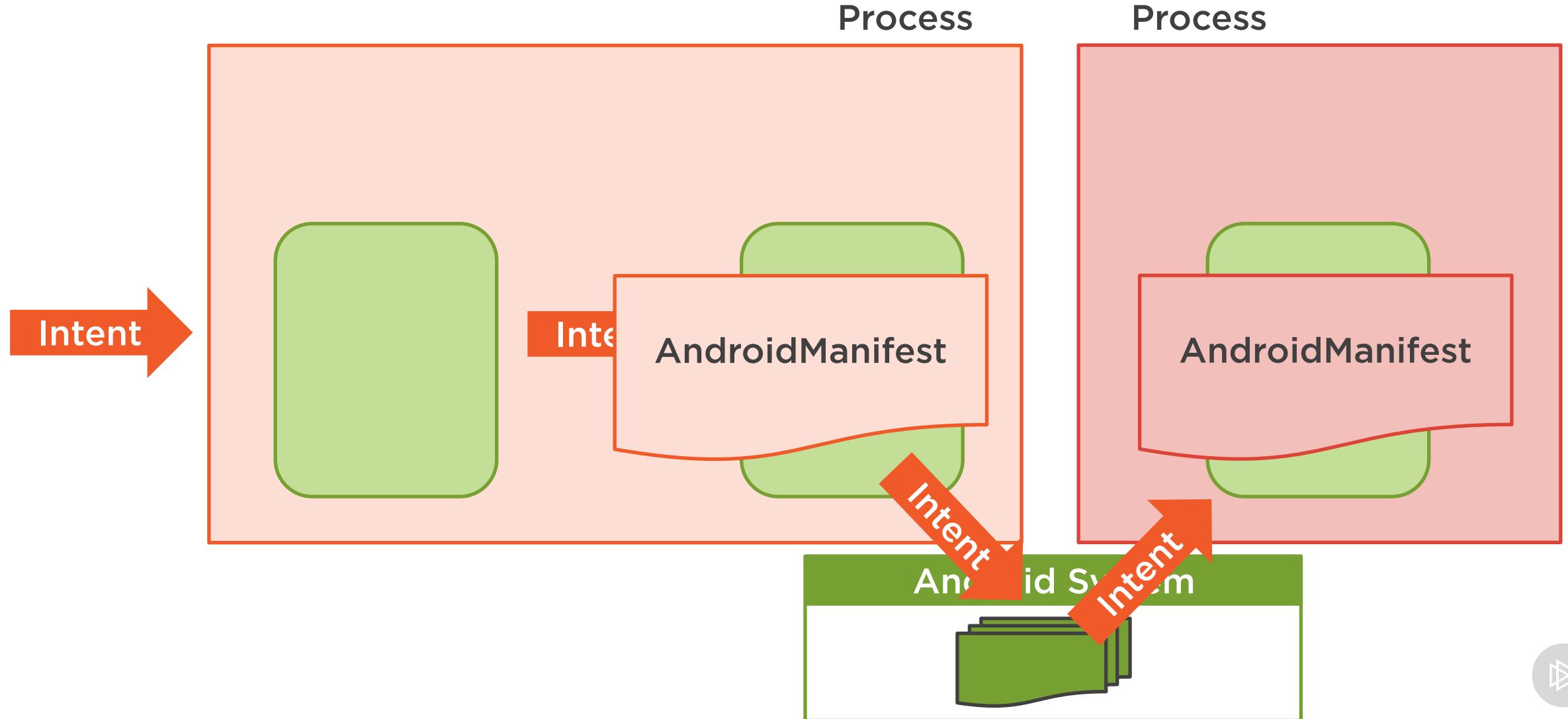
- Use Intent.setDataAndType
- setType and setData cancel each other



# Implicit Intents



# Implicit Intents



# Activities with Results

**Some Activity classes return results**

## **Camera Activity**

- Presents camera functionality
- Returns image thumbnail

## **Contact Activity**

- Presents contact UI
- Returns selected contact info

**Many others**



# Activities with Results

**Started differently than other activities**

- Use `startActivityForResult`

**Parameters passed to `startActivityForResult`**

- Intent
- App defined integer identifier
  - Differentiates results within your app



# Activities with Results

## Receiving results

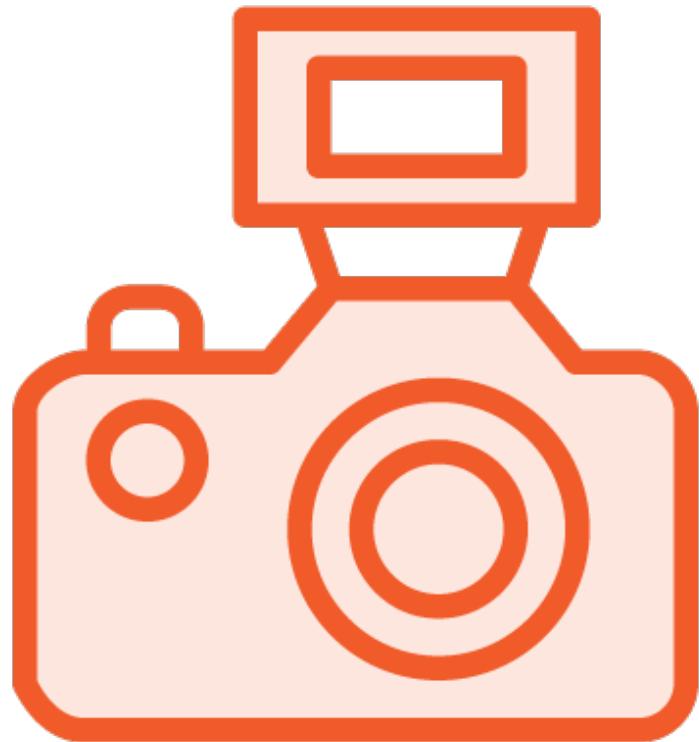
- Override your Activity's `onActivityResult`

## Parameters received by `onActivityResult`

- App defined integer identifier
- Result code
  - `RESULT_OK` indicates success
- Intent
  - Contains activity results



# Activity with Result Example

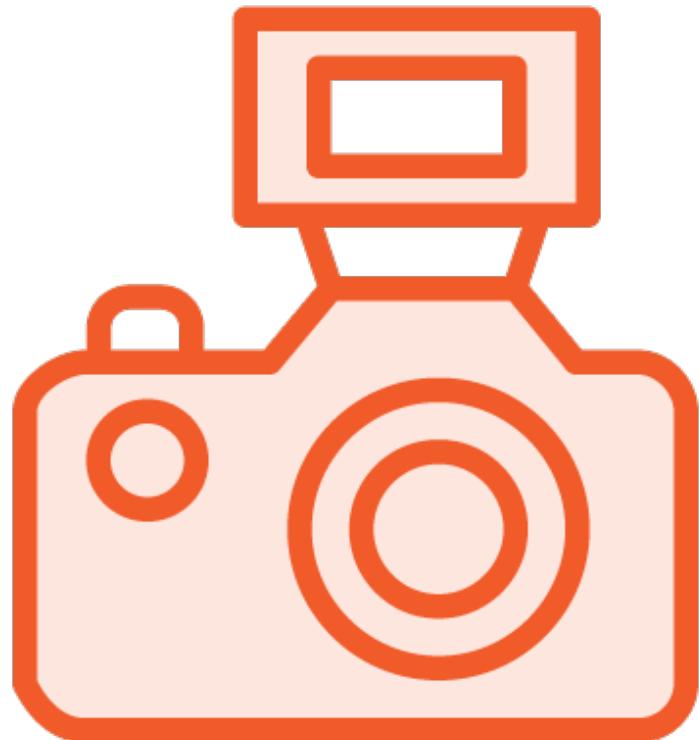


## Camera

- Presents Camera UI
- Stores full quality image in a file
- Returns image thumbnail as a result



# Starting the Activity



## Intent action

- `MediaStore.ACTION_IMAGE_CAPTURE`

## Extra

- `MediaStore.EXTRA_OUTPUT`
- File in which to save full quality image

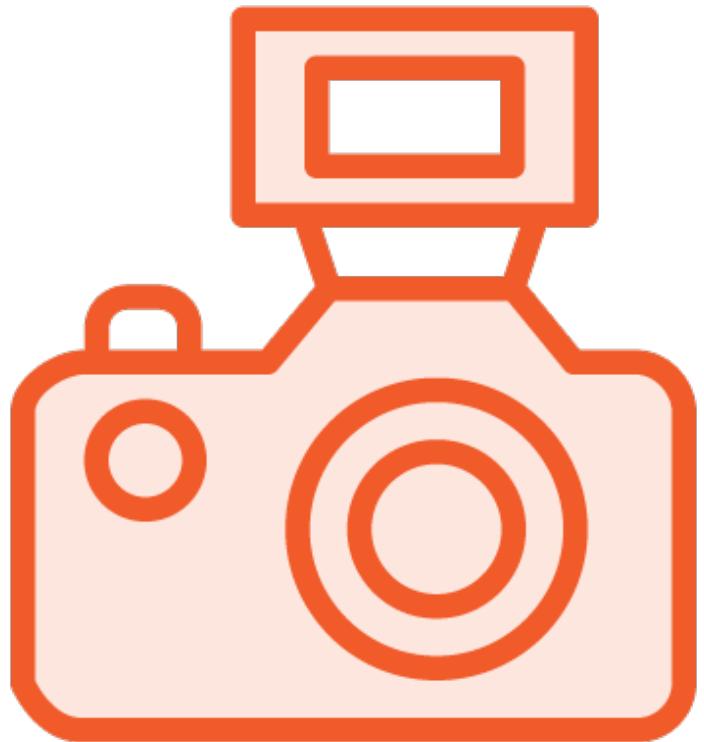


# Starting the Activity

```
public class MyActivity extends AppCompatActivity {  
    private static final int SHOW_CAMERA = 1;  
  
    private void showCamera(Uri photoFile) {  
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
        intent.putExtra(MediaStore.EXTRA_OUTPUT, photoFile);  
        startActivityForResult(intent, SHOW_CAMERA)  
    }  
  
    // other members elided for clarity  
}
```



# Receiving the Result



**Check for request code of SHOW\_CAMERA**

- Identifies the result is for our request

**Check for result code of RESULT\_OK**

- Indicates success
- Full quality image stored in file

**Retrieve thumbnail**

- Stored in result intent as a bitmap
- Name is “data”



# Receiving the Result

```
public class MyActivity extends AppCompatActivity {  
    private static final int SHOW_CAMERA = 1;  
    @Override  
    protected void onActivityResult(  
        int requestCode, int resultCode, Intent result) {  
        if(requestCode == SHOW_CAMERA && resultCode == RESULT_OK) {  
            Bitmap thumbnail = result.getParcelable("data");  
            // Do something  
        }  
    }  
    // other members elided for clarity  
}
```



# Application Experience

## Generally composed of multiple Activities

- Most probably come from your app
- Others may come from other apps
- Android needs to manage this flow
- Flow is managed as a task



# What is a Task?

A task is a collection of activities that users interact with when performing a certain job.



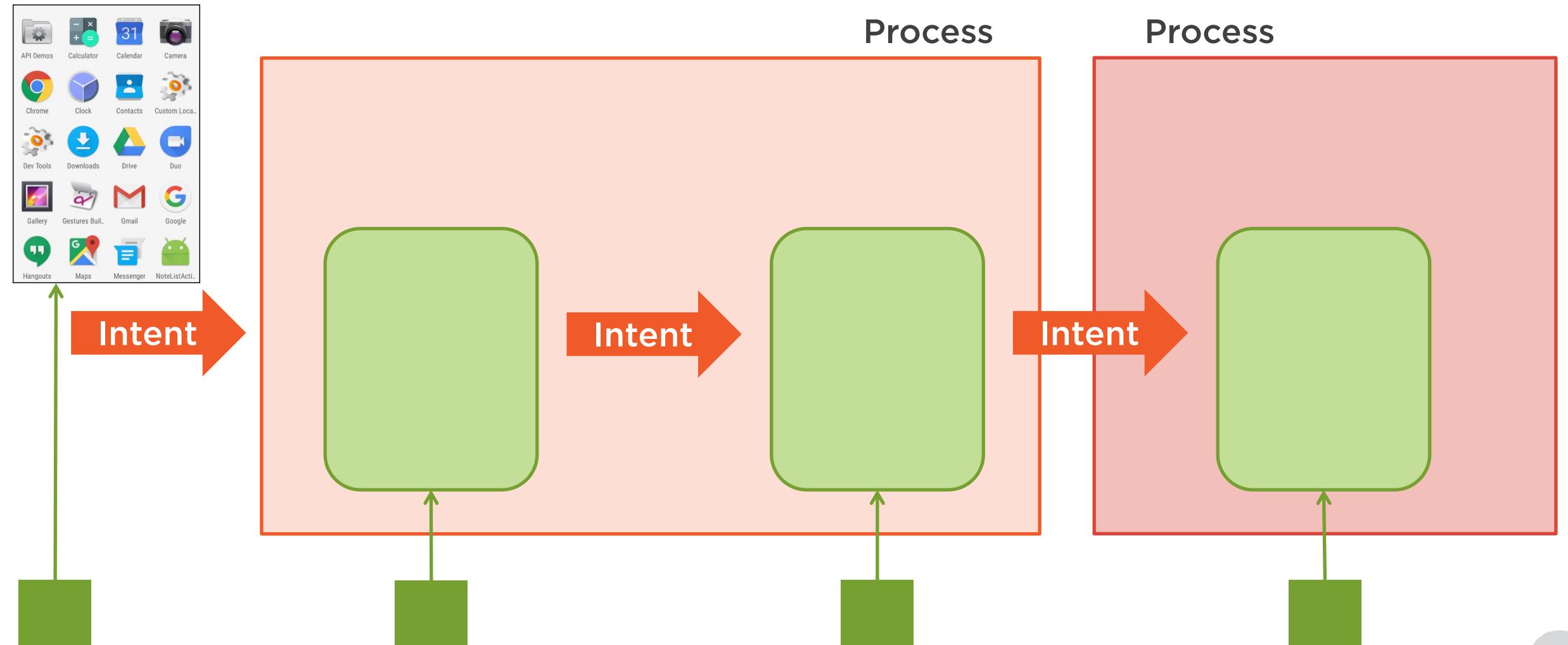
# Application Experience

## Task

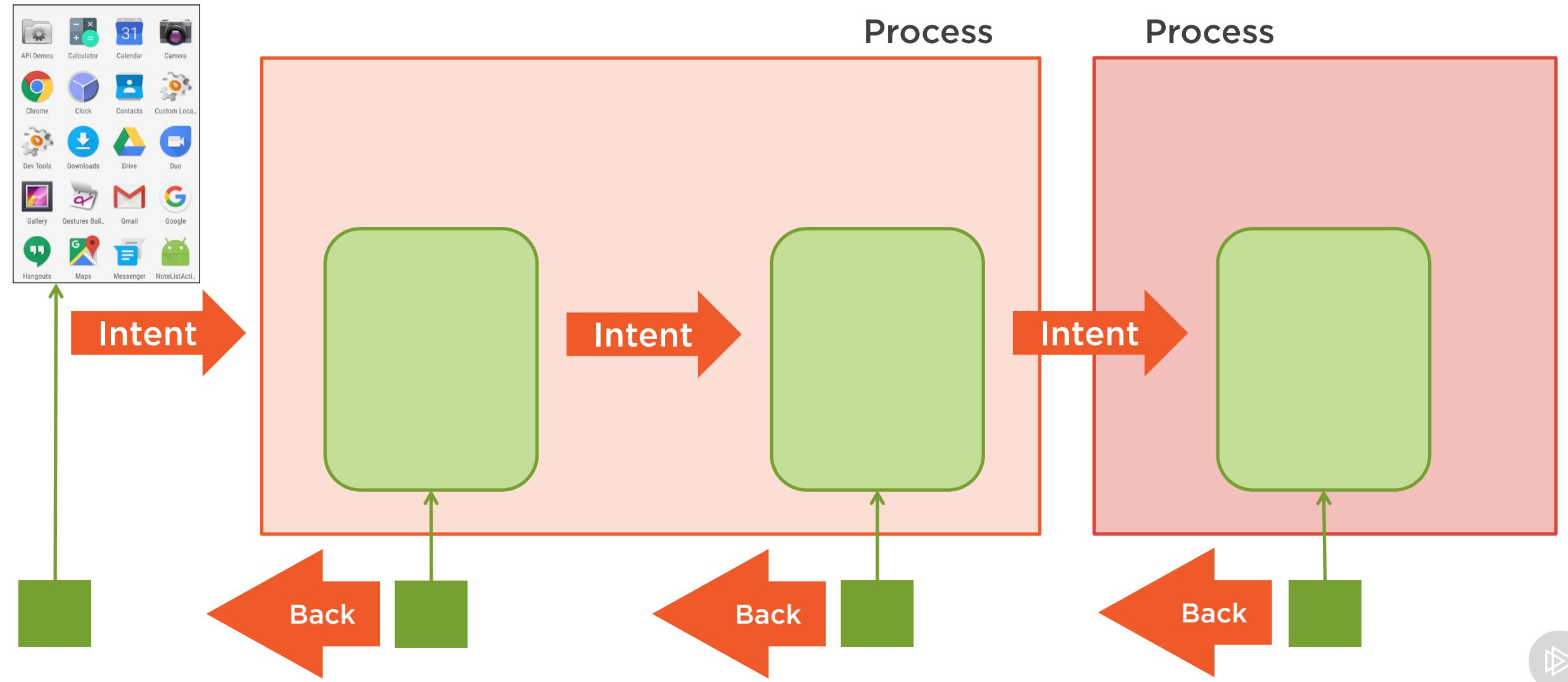
- Managed as a stack
  - Known as the back stack
- Activities added going forward
- Back button removes Activities
  - Causes Activity to be destroyed



# Task



# Task



# Application Experience

## Managing persistent state

- Use edit-in-place model
- Changes saved with no special action

## Saving changes

- Write to backing store when leaving
- Handle in onPause

## Handling new entries

- New entries created right away
- Handle in onCreate



# Activity Lifecycle

## Common causes of Activity destruction

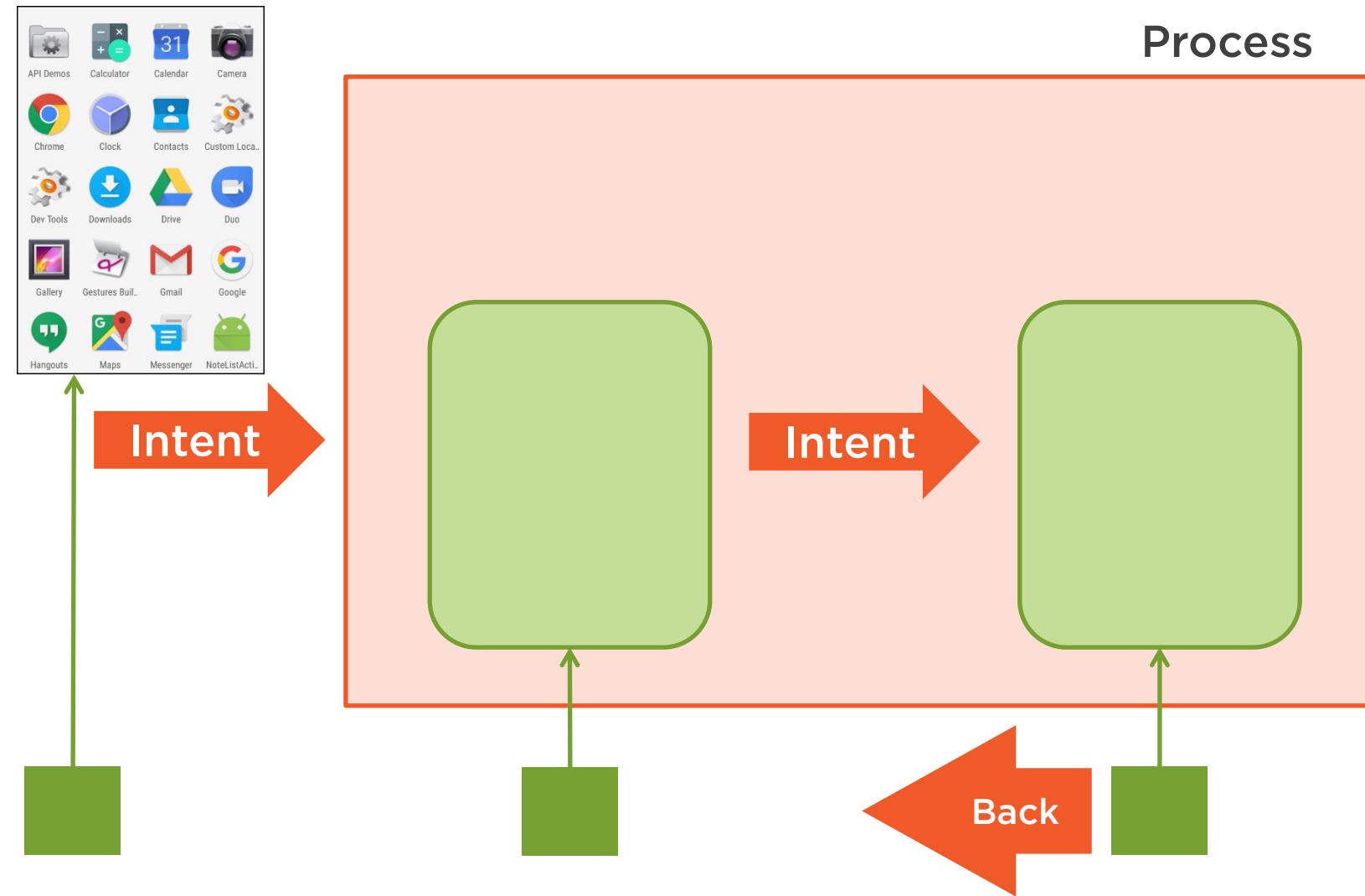
- Leaving with the back button
- Calling finish method
- System initiated

## System initiated destruction

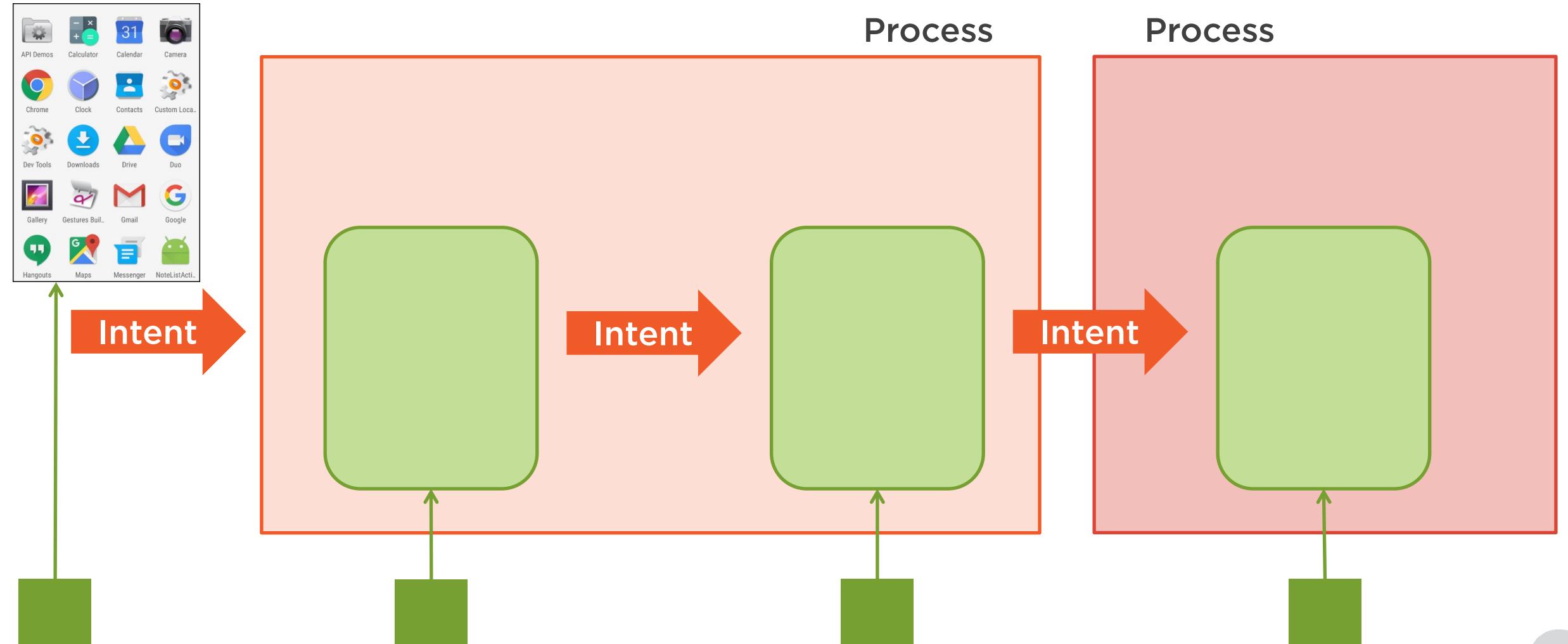
- Generally to reclaim resources
- Prolonged period in the background
- System experiencing resource pressure



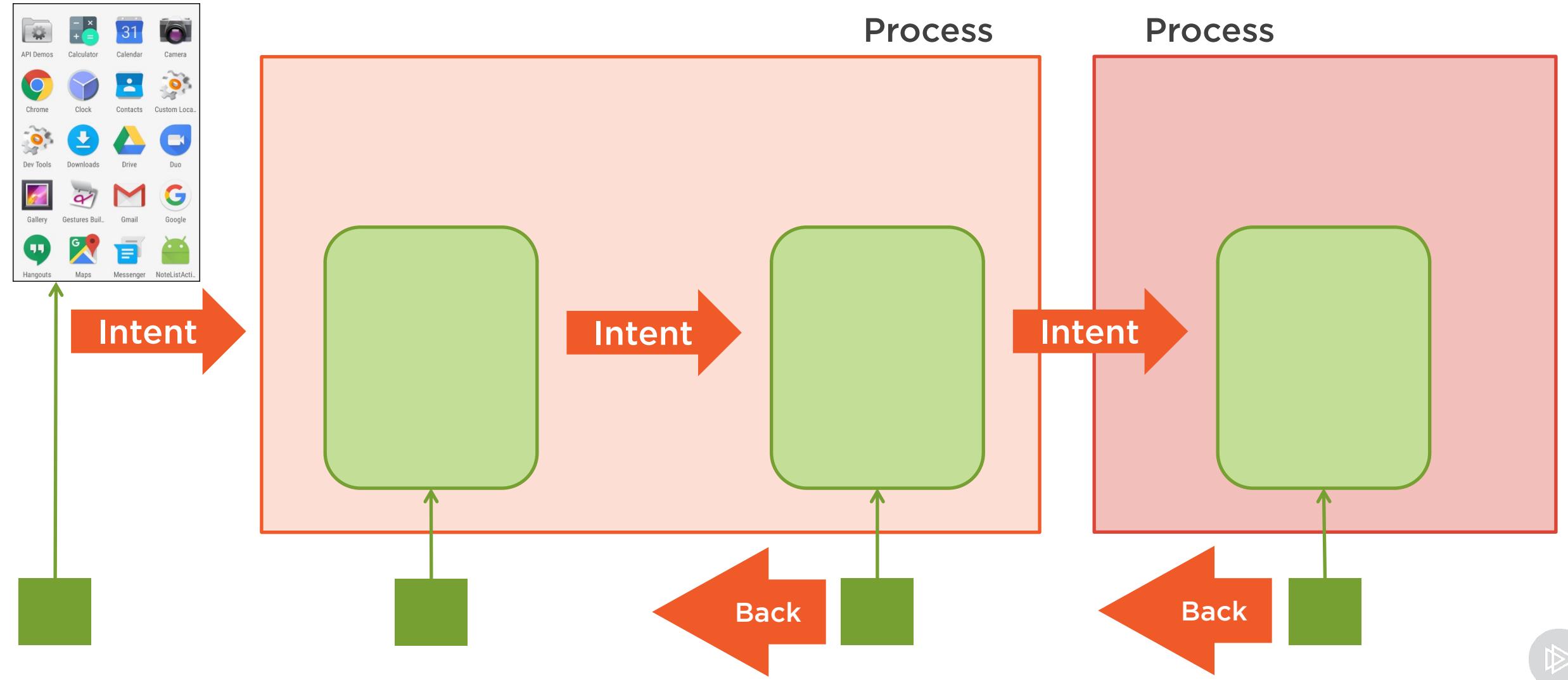
# Task



# Task



# Task



# Activity Lifecycle Methods

## Lifetimes within Activity lifecycle

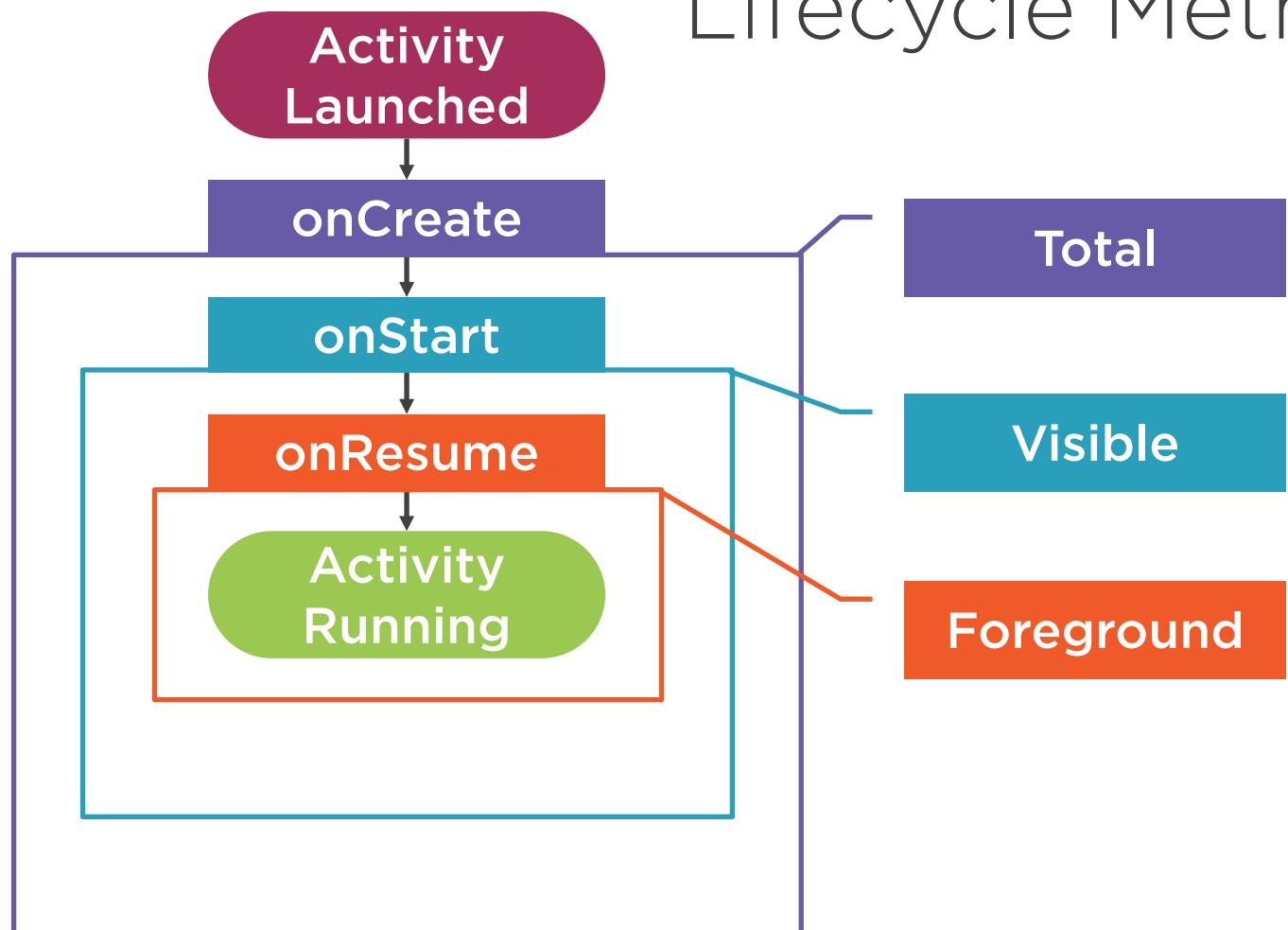
- Total lifetime
- Visible lifetime
- Foreground lifetime

## Activity lifecycle methods

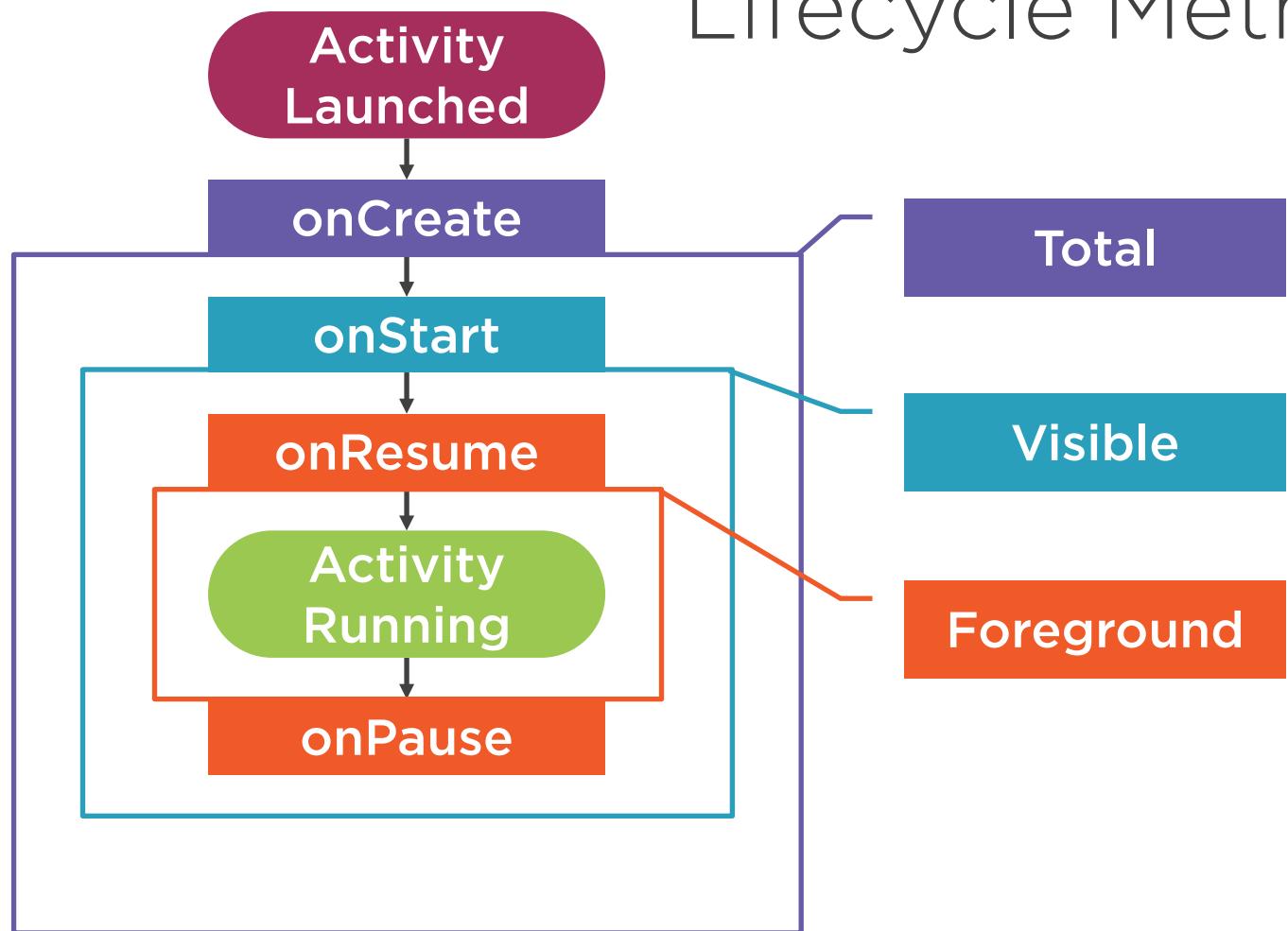
- Methods for start/end of each lifetime
- A few additional methods for transitions



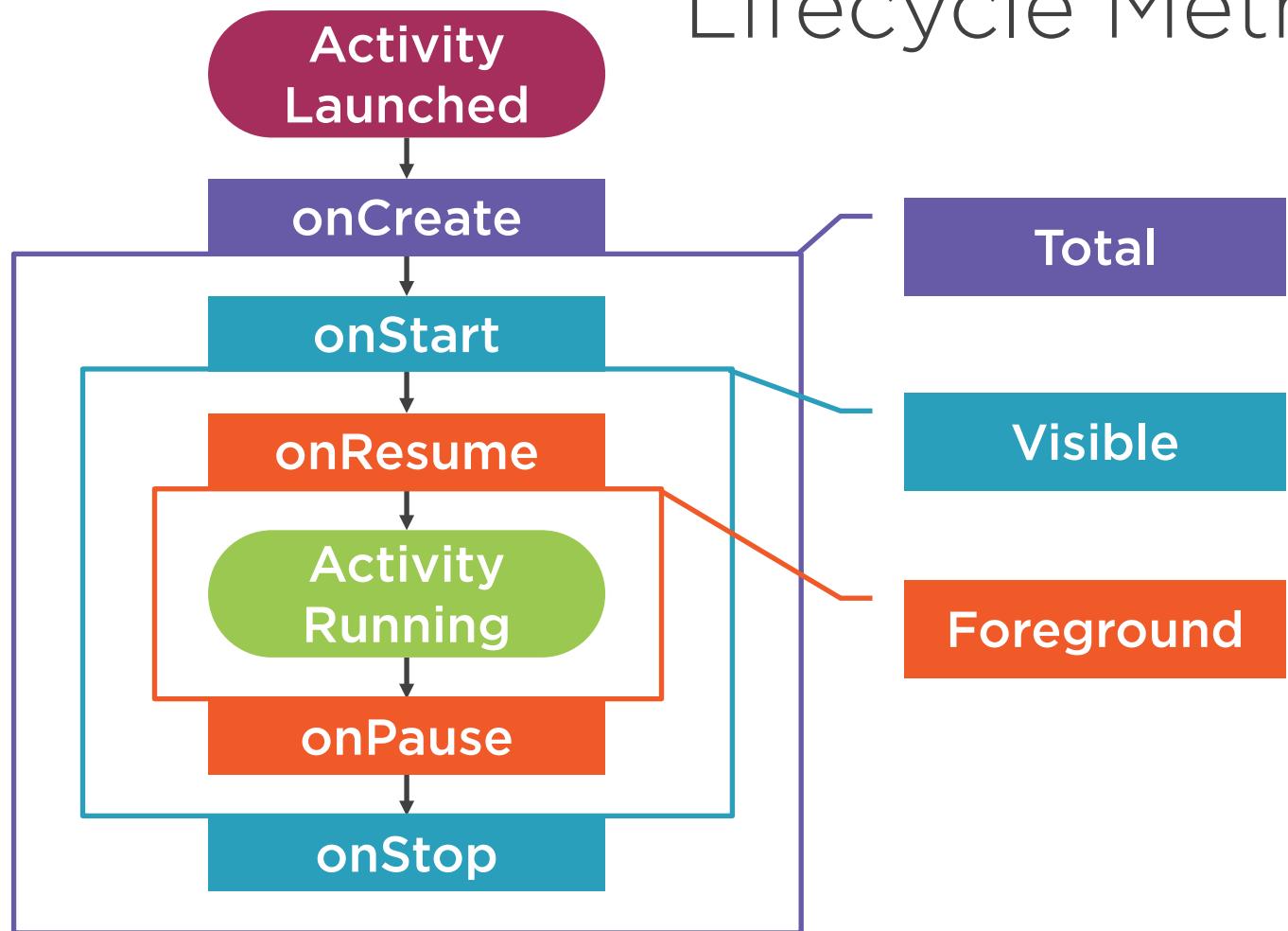
# Lifecycle Methods



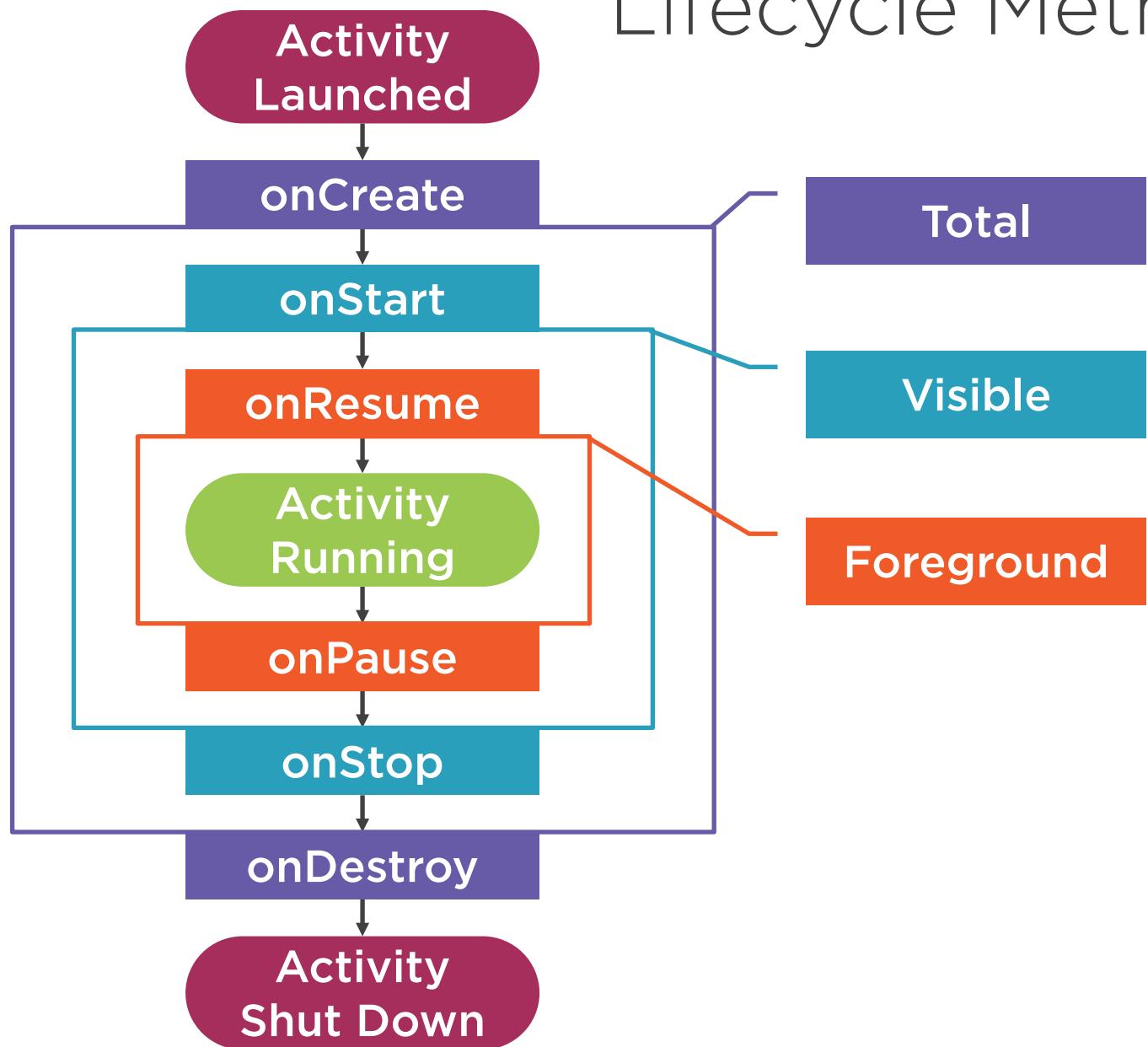
# Lifecycle Methods



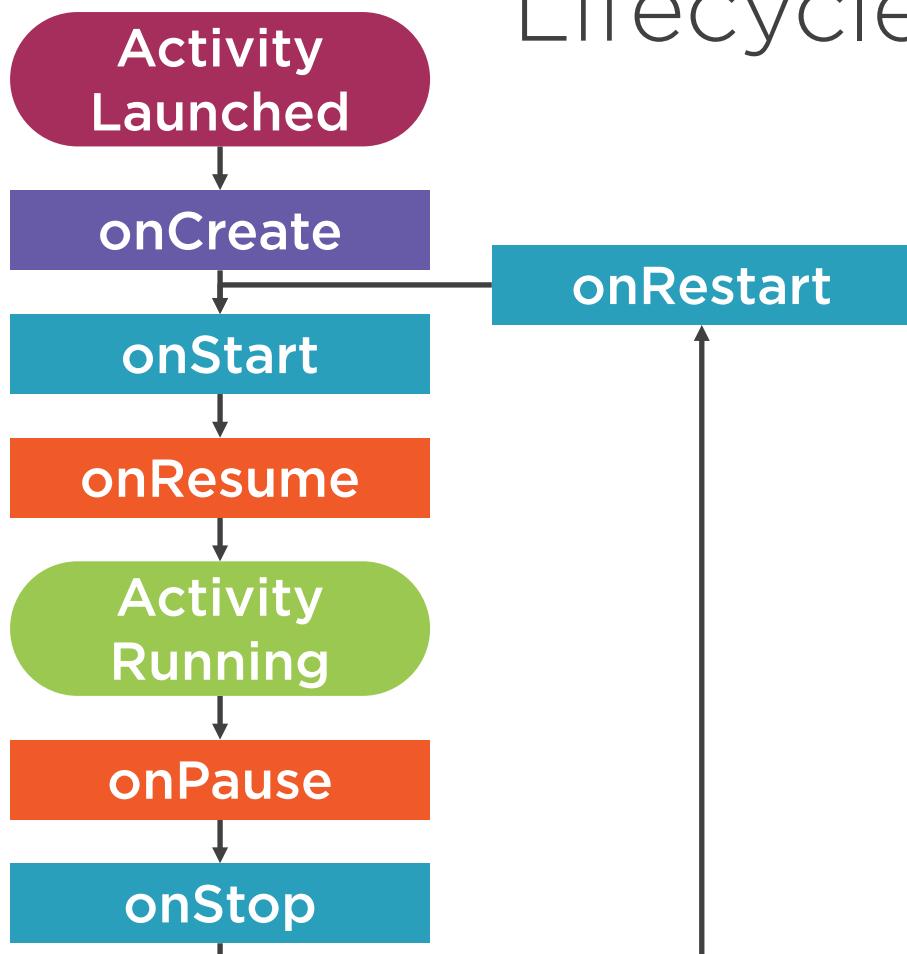
# Lifecycle Methods



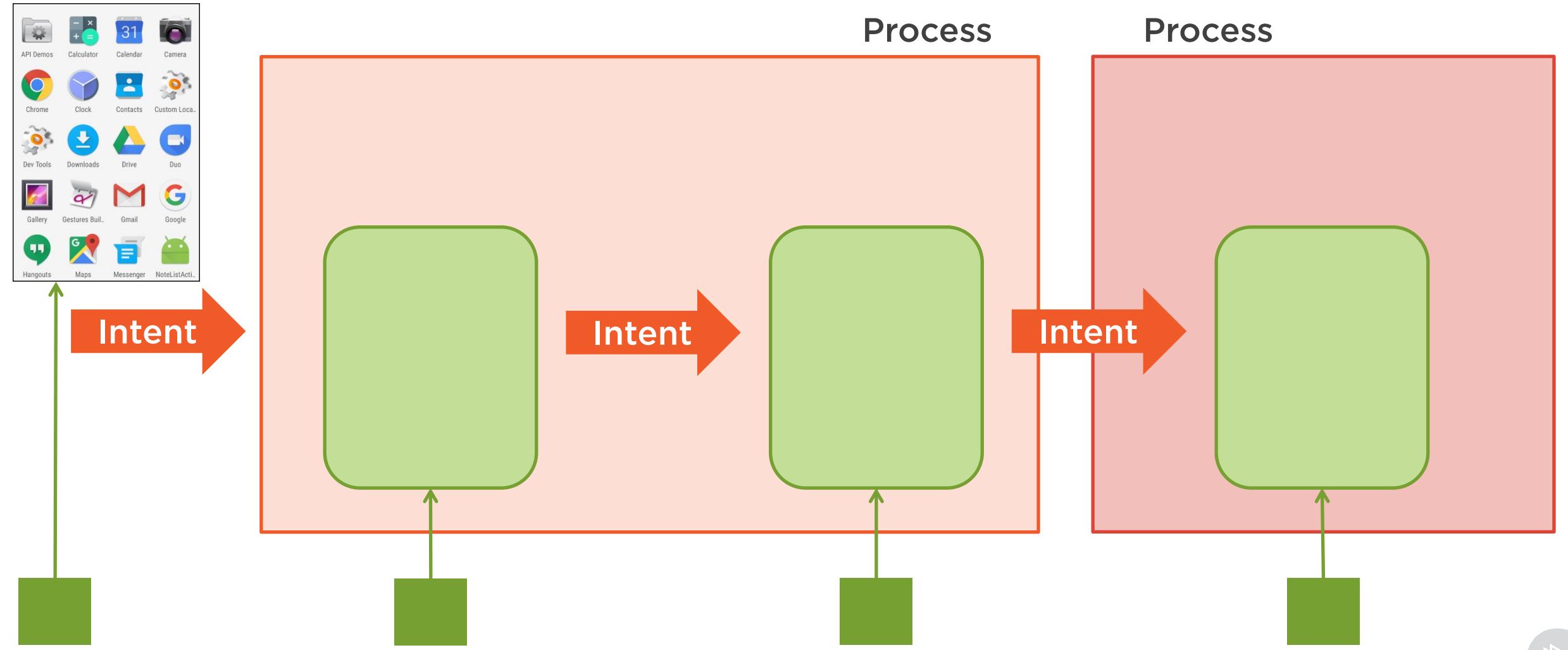
# Lifecycle Methods



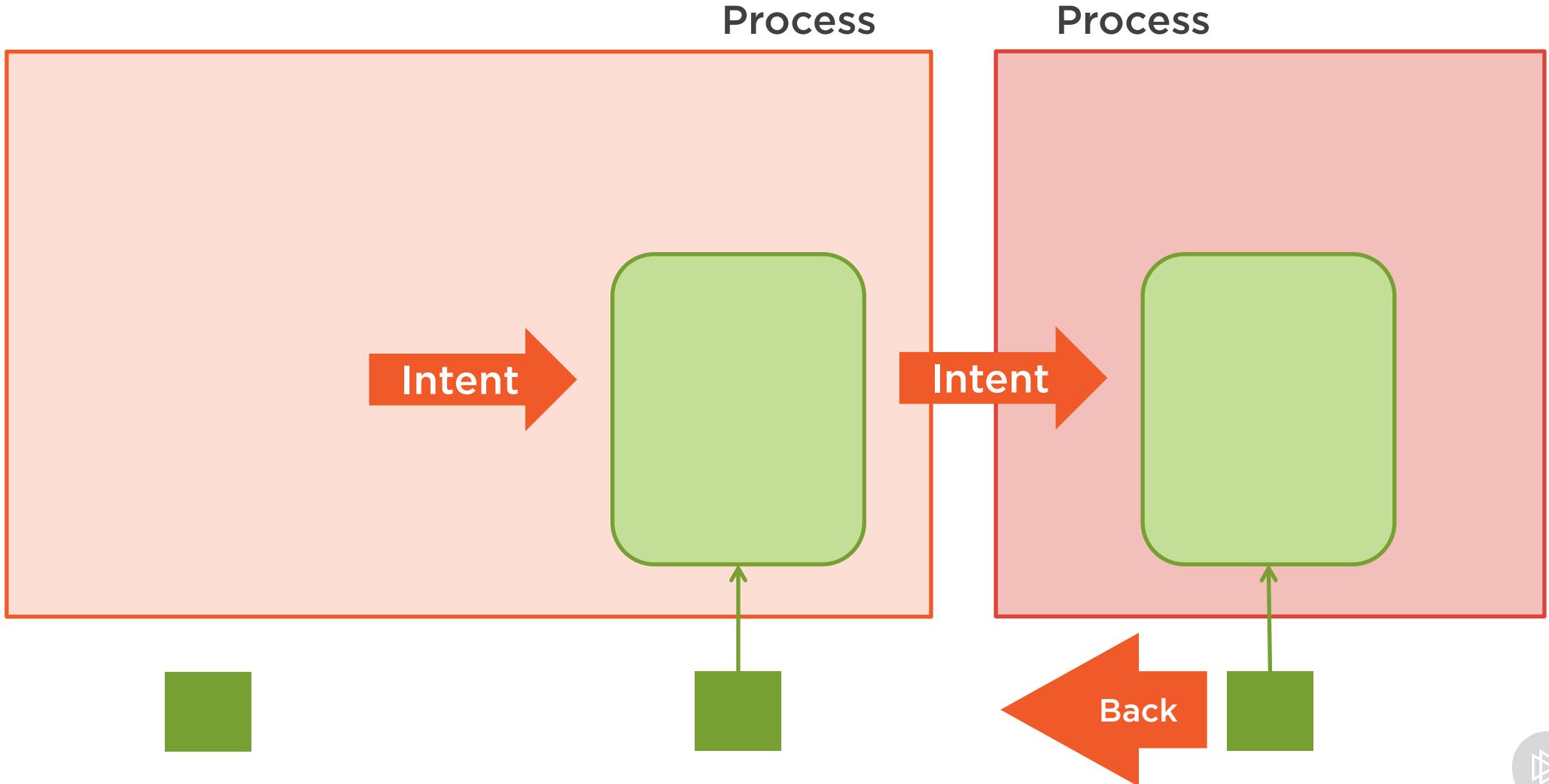
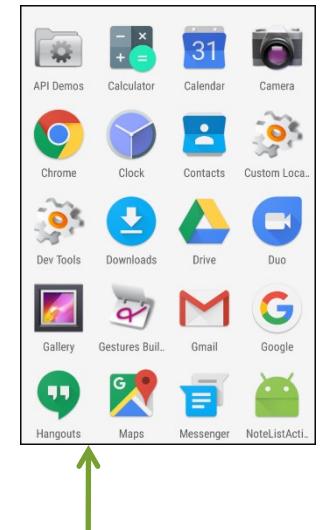
# Lifecycle Methods



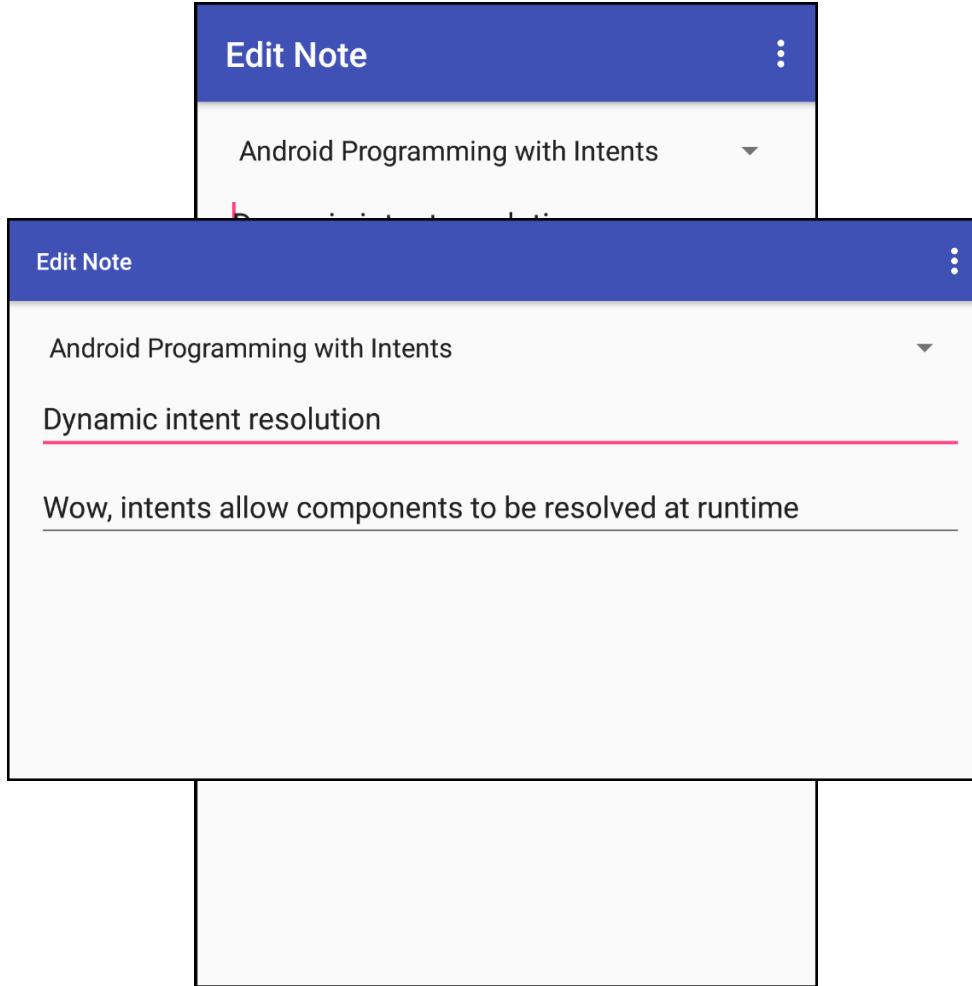
# Activity State Management



# Activity State Management



# Activity State Management



# Activity State Management

## Activities provide state management

- Opportunity to save before destroy
- Saved state provided on restore

### Saving state

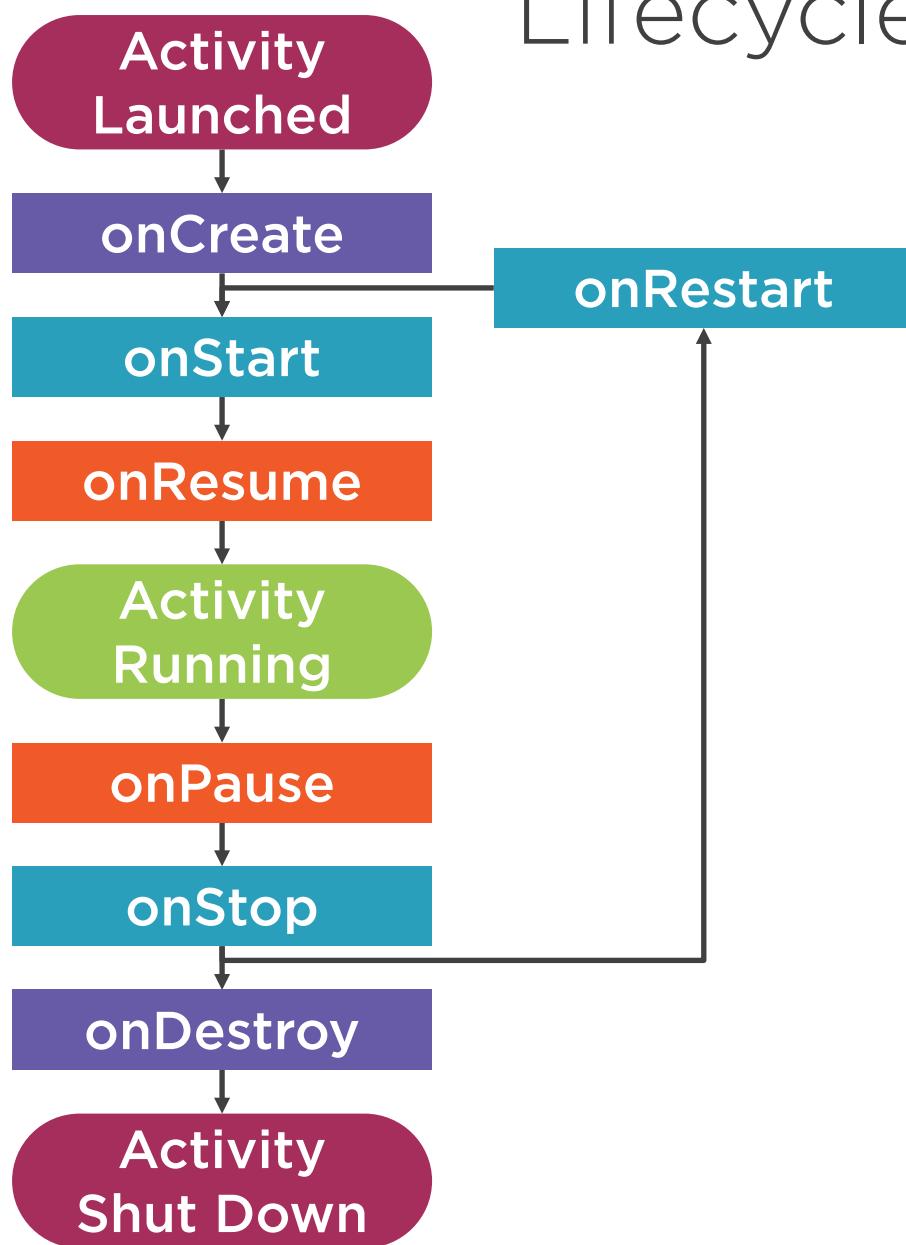
- `onSaveInstanceState`
- Write Activity state to passed Bundle

### Restoring state

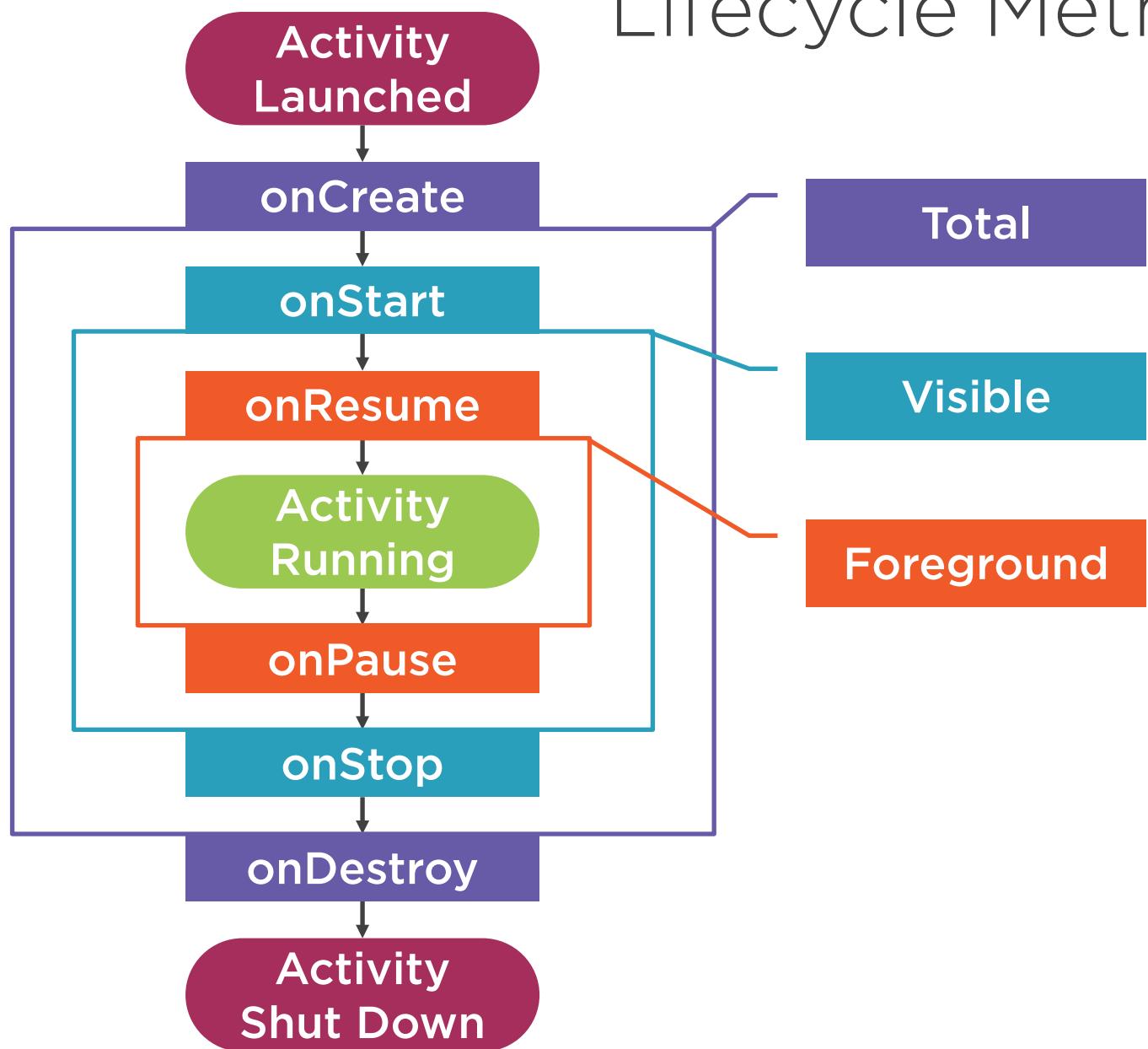
- `onCreate`
- Receives saved Bundle on restore
- Bundle is null on initial create
- Intent remains available on restore



# Lifecycle Methods



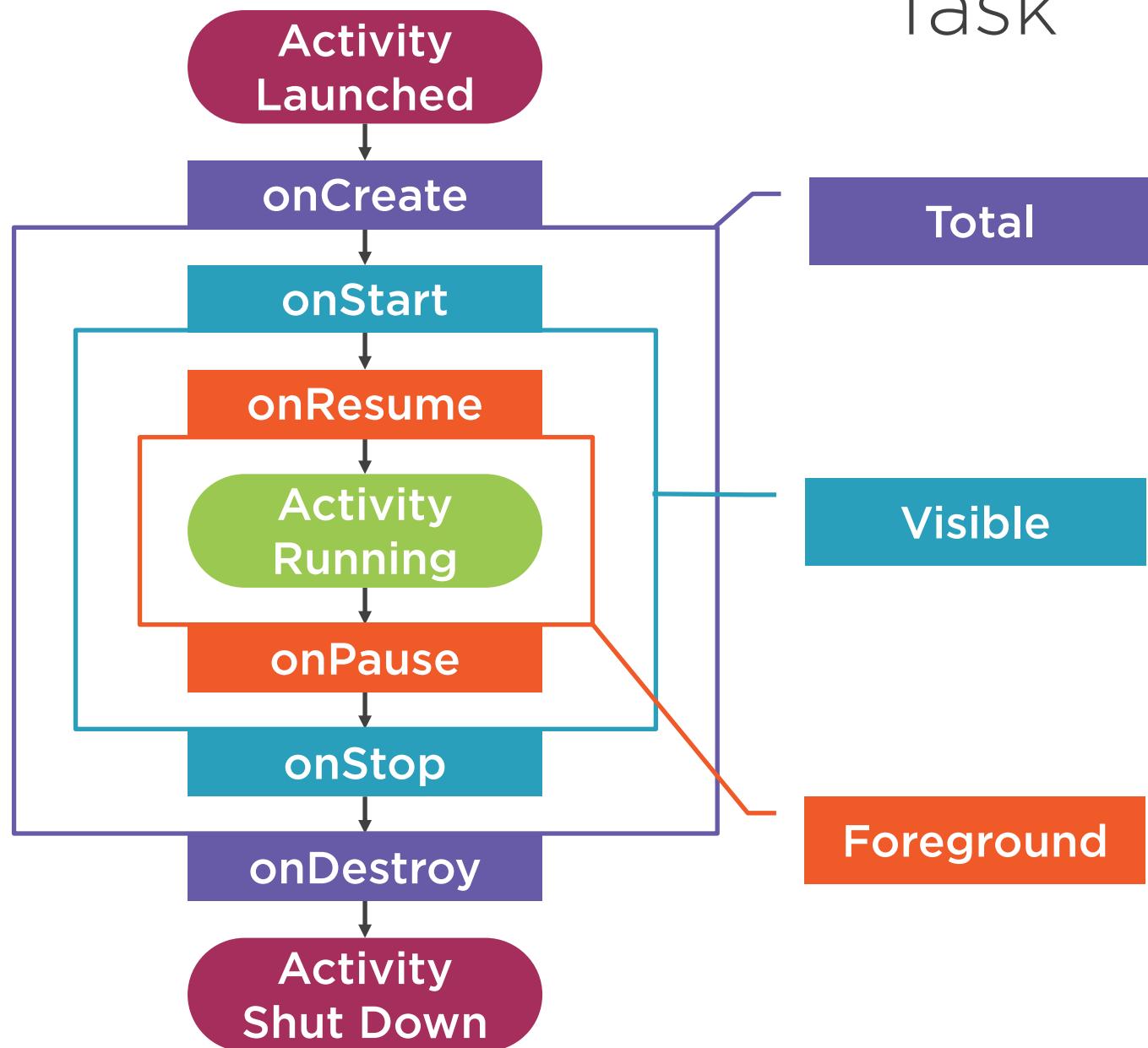
# Lifecycle Methods



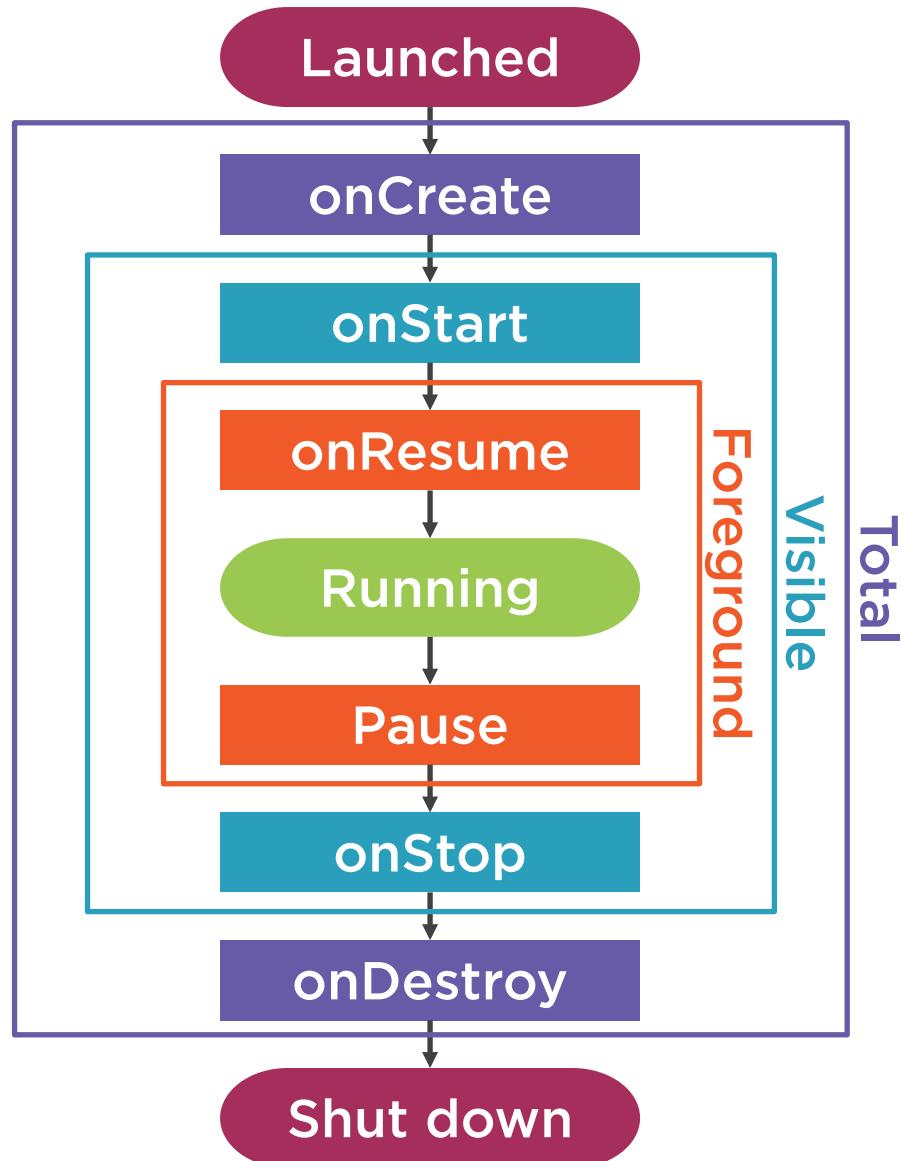
# Summary



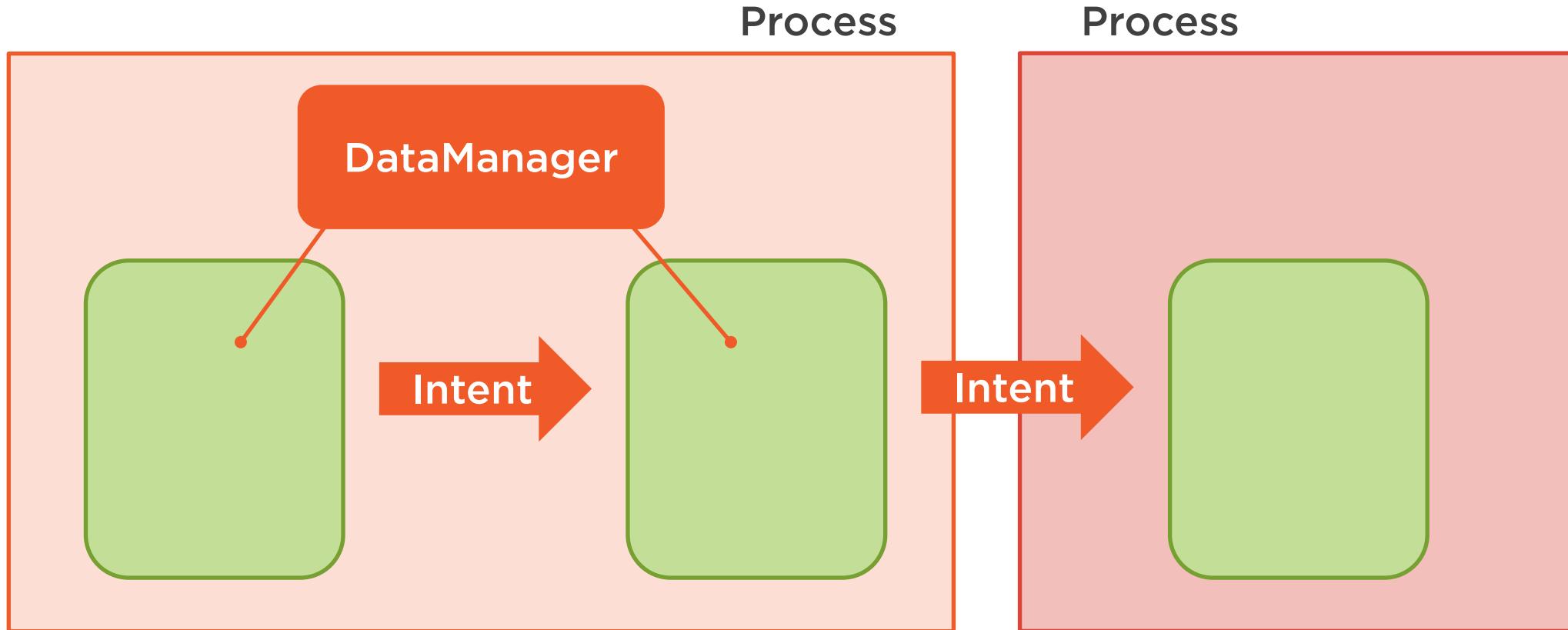
# Task



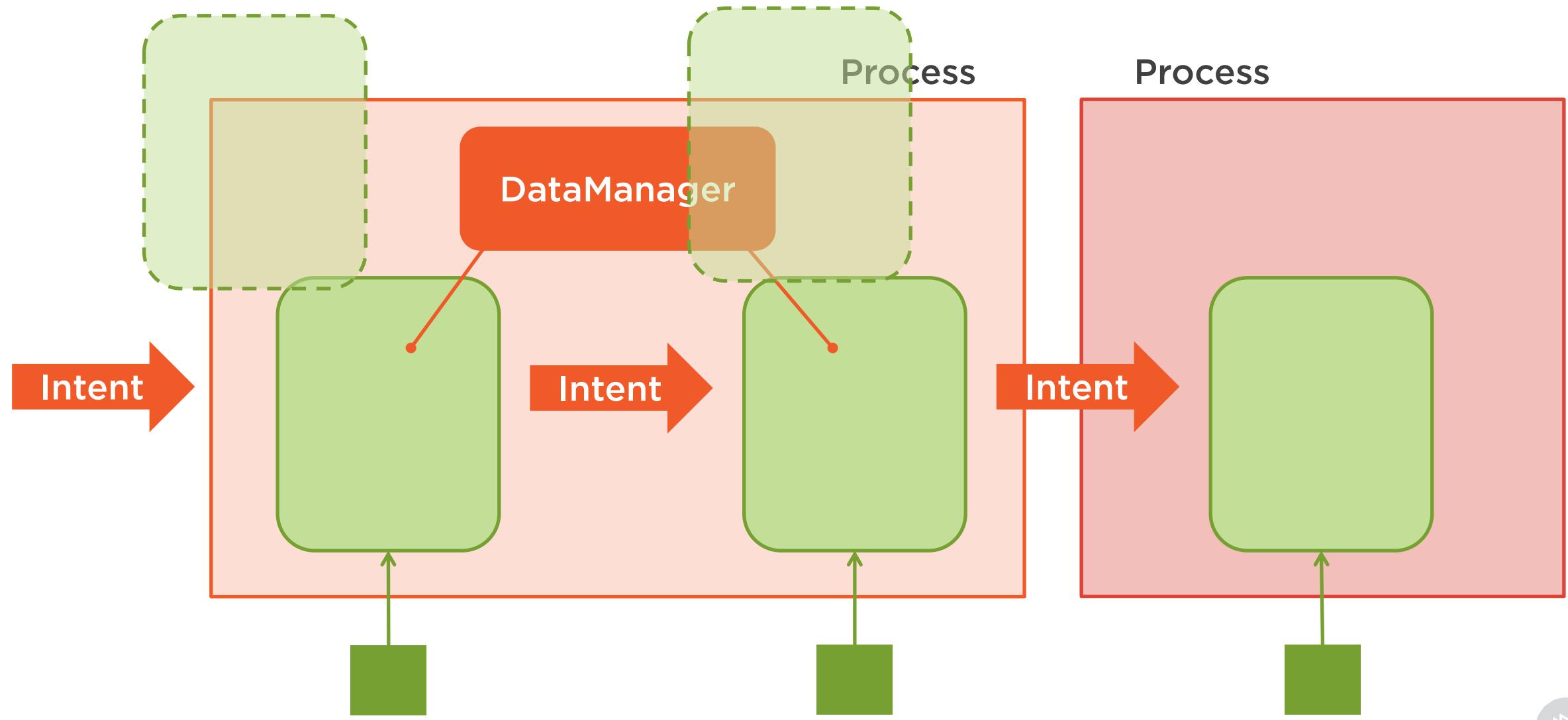
# Task



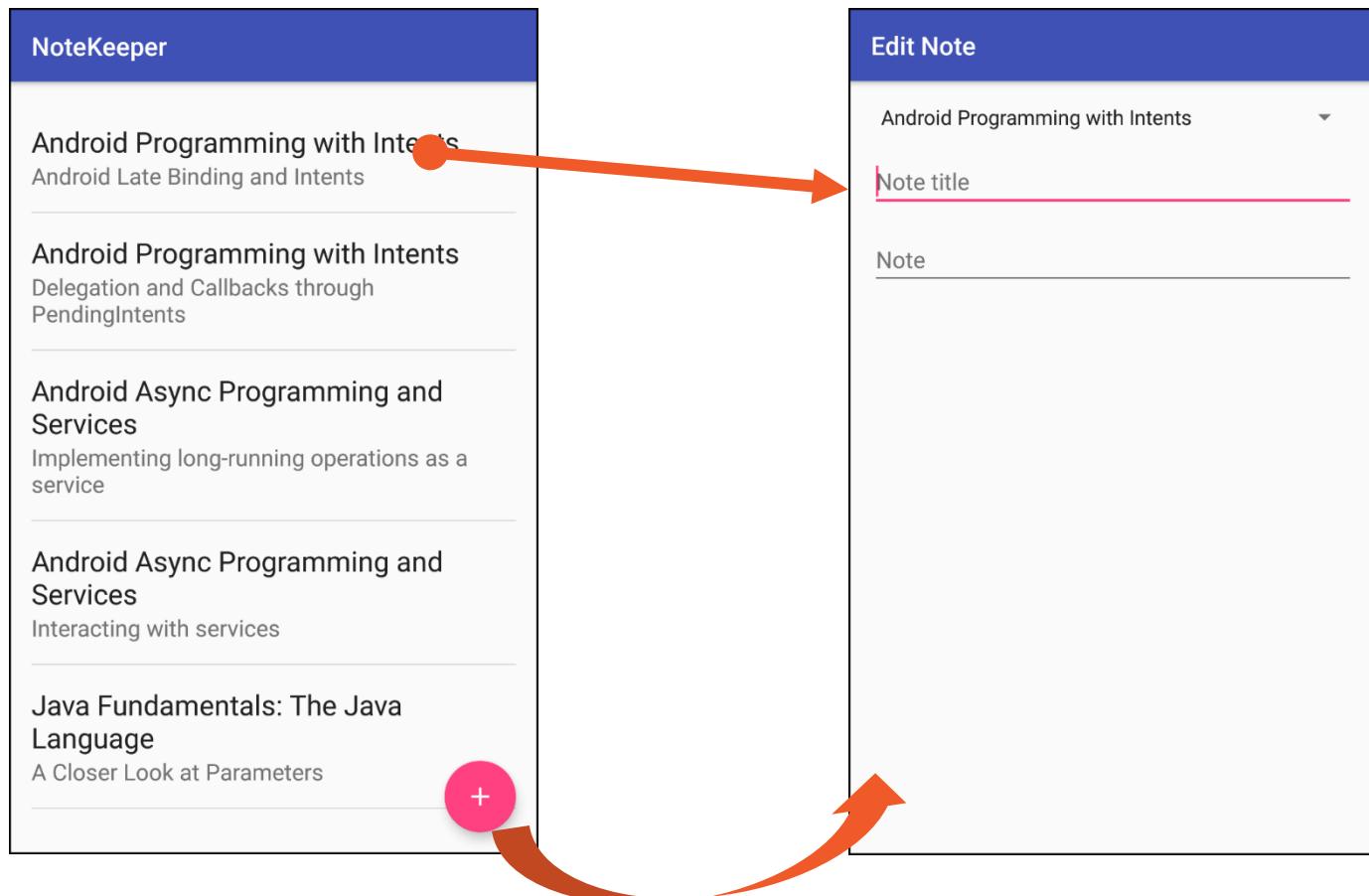
# Activity Interaction



# Activity Interaction



# A Quick Reminder of Our App



# Activity Interaction

**Android is a component-oriented platform**

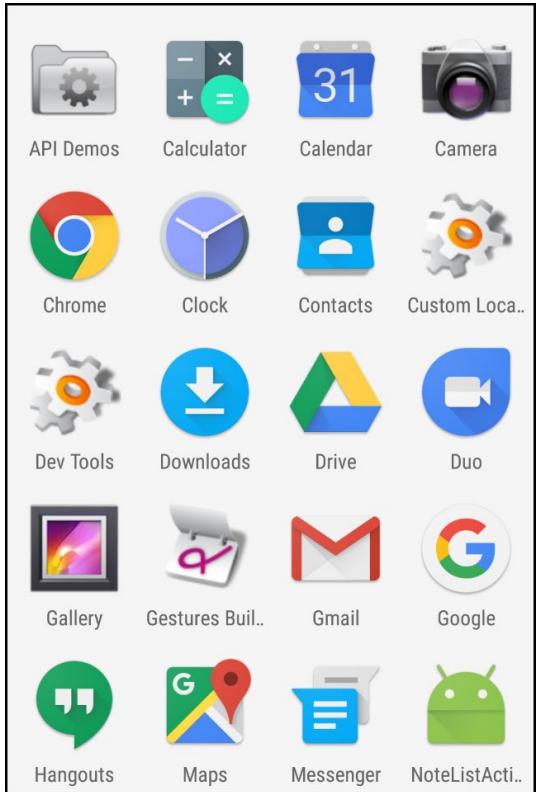
- A number of different types
- Activities are the most familiar

**Activities are distinct from one another**

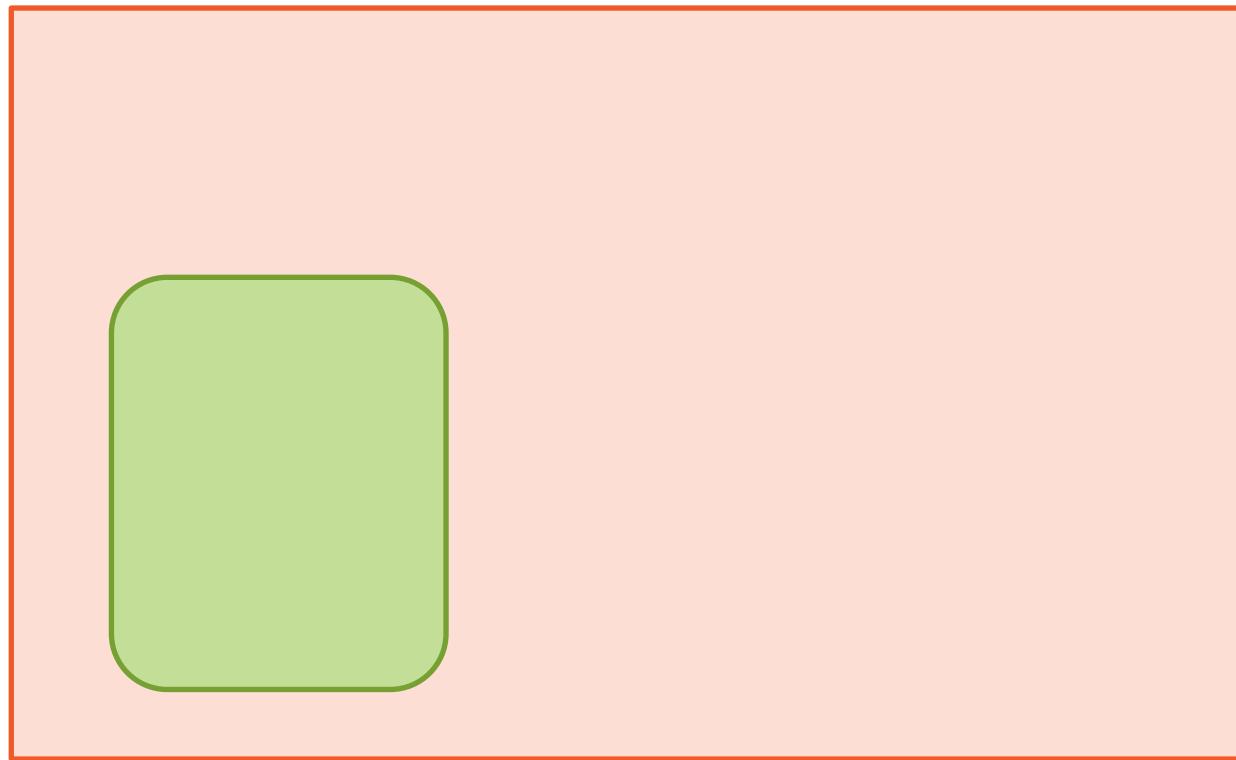
- One cannot directly create another
- Rely on intents to interact



# Activity Interaction



Intent →



# Starting an Activity Within Your App

## Create an intent

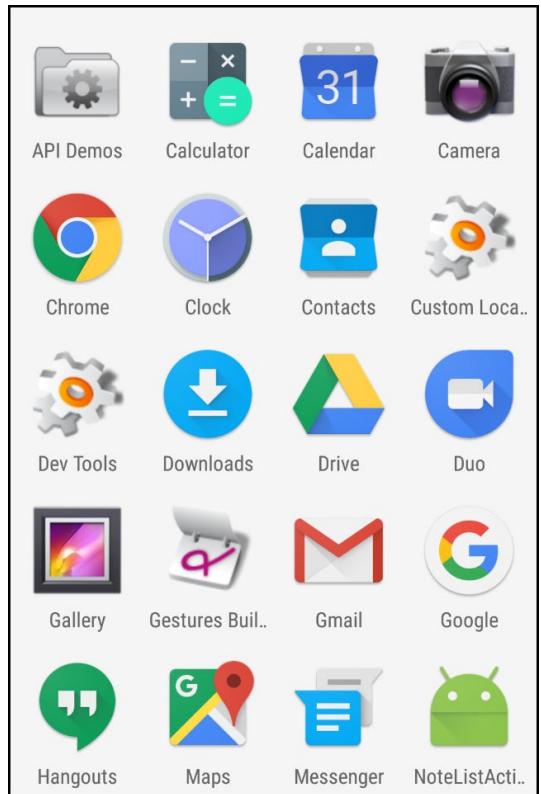
- Identifies the desired Activity
- Often can just be Activity class info

## Call startActivity

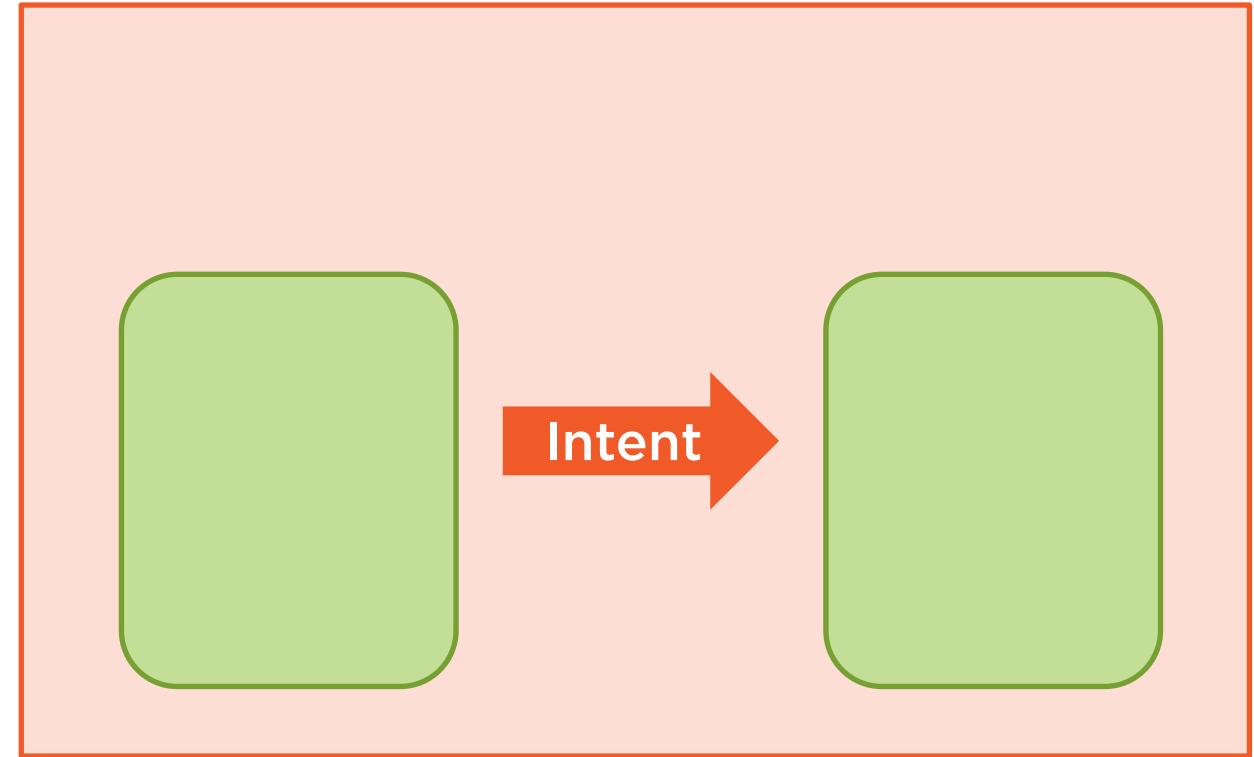
- Pass the intent
- Launches Activity matching the intent



# Activity Interaction



Intent



Process



# Describing Operations with Intents

## **Intents describe a desired operation**

- Often need more than just a target
- May need to provide additional info

## **Intent extras provide additional info**

- Name value pairs
- Names & values are operation-defined
- Added to intent with putExtra overloads



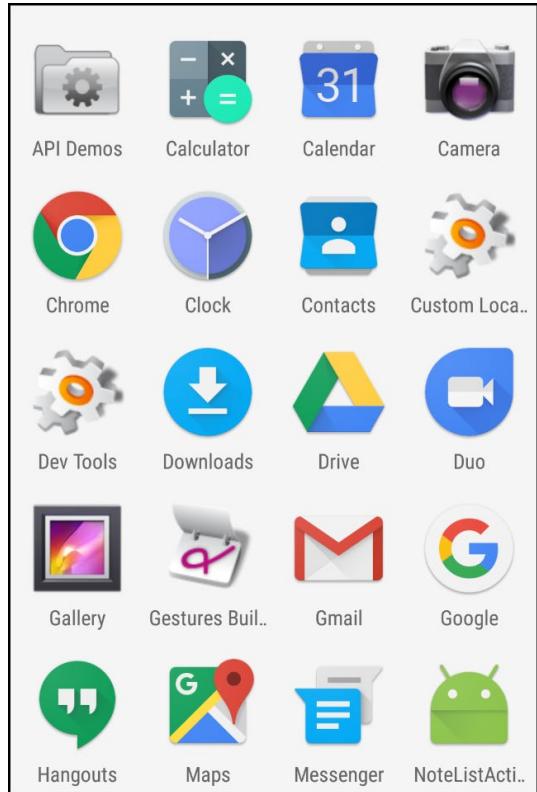
# Describing Operations with Intents

## Accessing intent info

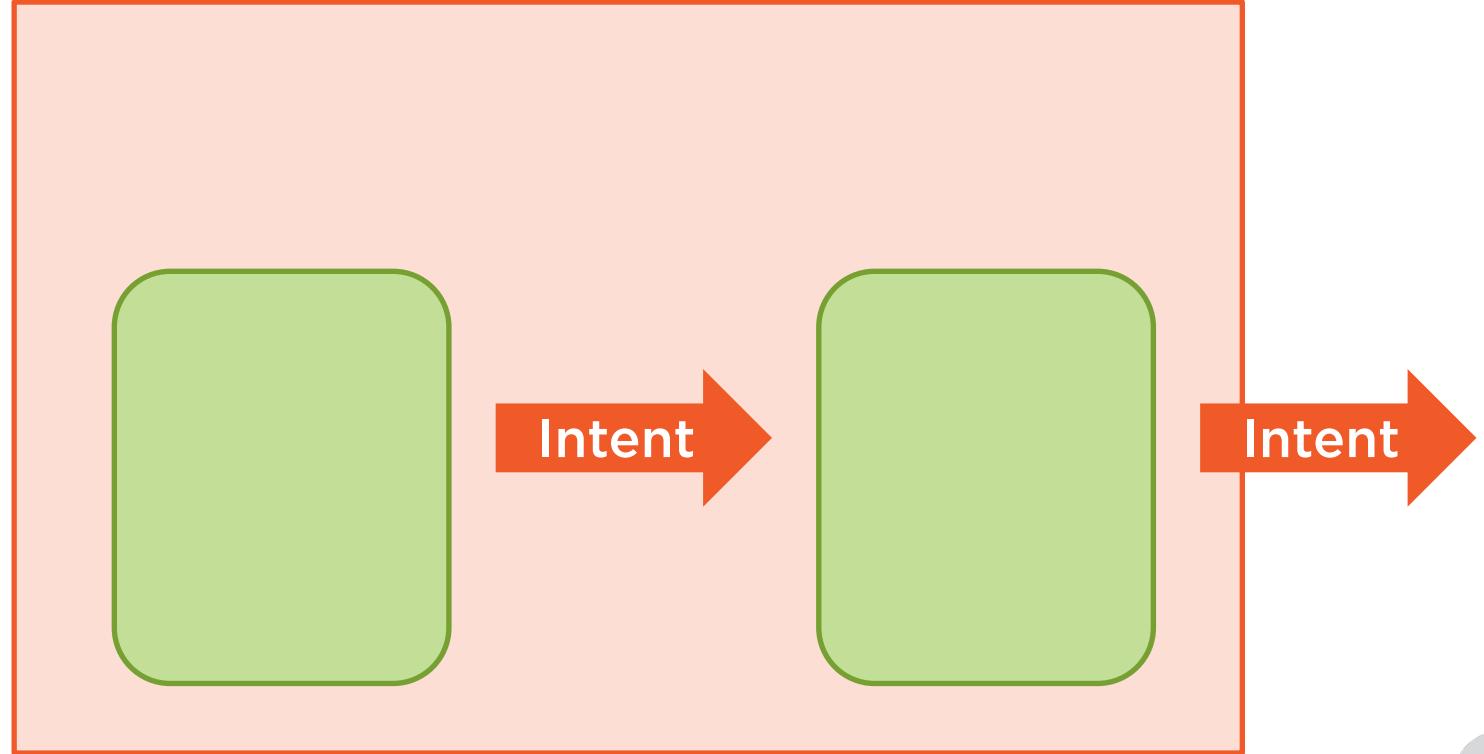
- Activity can access intent that started it
- Use getIntent
- Use Intent.getXXExtra to retrieve extras
  - Method names include return type



# Describing Operations with Intents



Intent →



# Describing Operations with Intents

**Intents must be cross-process friendly**

- Limits allowable extras

**Supported extra types**

- Primitive types and String
- Arrays of supported types
- Some ArrayLists
- A few other special types

**Most reference types not directly supported**

- Require special handling



# Reference Types as Intent Extras

**Reference types need to be “flattened”**

- Converted to a bunch of bytes

**One option is Java serialization**

- Supported but not preferred
- Serialization is very runtime expensive

**Use Parcelable API**

- Much more efficient than serialization
- Must be explicitly implemented



# Reference Types as Intent Extras

## Implement Parcelable interface

- `describeContents`
  - Indicates special behaviors
  - Generally can return 0
- `writeToParcel`
  - Receives a Parcel instance
  - Use `Parcel.writeXX` to store content



# Reference Types as Intent Extras

**Provide public static final CREATOR field**

- Is a `Parcelable.Creator` implementation

**`Parcelable.Creator` interface**

- `createFromParcel`
  - Responsible to create new type instance
  - Receives a `Parcel` instance
  - Use `Parcel.readXX` to access content
- `newArray`
  - Receives a size
  - Responsible to create array of type



## Show android image

- Launcher
- Say that launcher indicates desire to start Activity
- Android starts process containing activity
- Creates Activity





## Starting Activity

- Intents
- Implicit vs. Explicit



**Intents are used to activate components**

- Contains component identifying info

**Manifest helps launch**





## Demo

- Show manifest
- Create ListActivity
- Show NoteActivity





**Passing information with Intents**





**Parcalable**





## Understanding lifecycle events





## Demo

- Use diagram to show challenge created by passing Note
- Switch to passing as integer



## **Android is a component-oriented platform**

- A number of different types
- Activities are the most familiar

## **Components have a well-defined lifetime**

- Components drive process lifetime



# Traditional Process-oriented Model

```
java com.jwhh.cmdline.Main
```

Process

```
Class Main {  
    public static void main() {  
        // do some stuff  
    }  
}
```



# What is an Activity?

An activity is a single, focused thing that the user can do.



# A Little More Detail About Activities

## Serve as the place to present the UI

- Provide a window
- UI is built with View-derived classes

## Have a lifecycle

- More than just a “screen”
- Lifecycle calls a series of methods
- Use the onCreate method to initialize the Activity



# Activity UI

## View

- Basic building block of UI
- Drawing and event handling
- Many specialized classes available

## ViewGroup

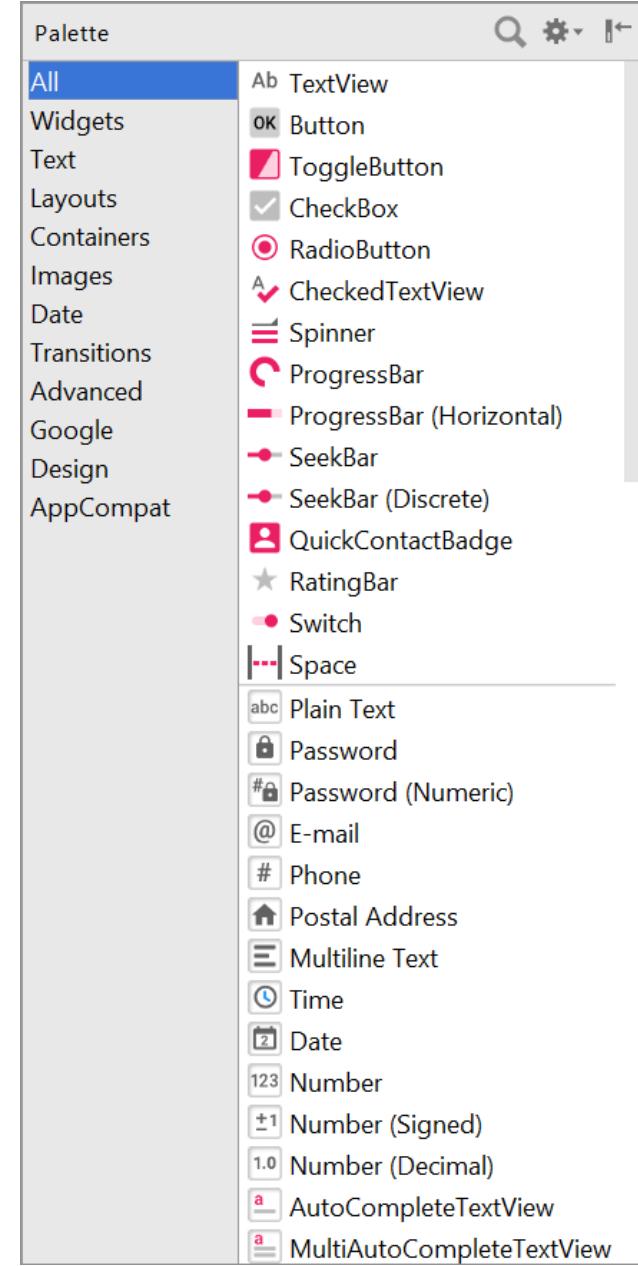
- Special View that holds other views

## Layout

- Special invisible ViewGroup
- Handle View positioning behavior
- Many specialized classes available



# Common View Classes



# Layout Classes

## Activity UIs need to be responsive

- Device display characteristics vary
- UI must adapt
- Absolute positioning would be limiting

## Layout classes provide positioning flexibility

- Arrange child Views
  - Children can include other layout classes
- Specific positioning behavior depends on the layout class



# Some Common Layout Classes

## FrameLayout

- Provides a blocked-out area
- Generally has only one direct child

## ScrollView

- Provides a scrollable area

## LinearLayout

- Horizontal or vertical arrangement
- Supports weighted distribution

## RelativeLayout

- Relative positioning
- Relative to one another or parent



# Simplifying Layout Creation

## Traditional layout classes have challenges

- UIs have become much richer

## Run-time challenges

- Sometimes have to nest layout classes
- Deep/complex nesting impacts speed

## Design-time challenges

- Achieving desired result with designer sometimes challenging
- In some cases end up working in the XML



# ConstraintLayout Class

## Extremely flexible layout class

- Often the only layout class needed

## First-class design-time experience

- Closely integrated with designer
- Rarely need to resort to XML



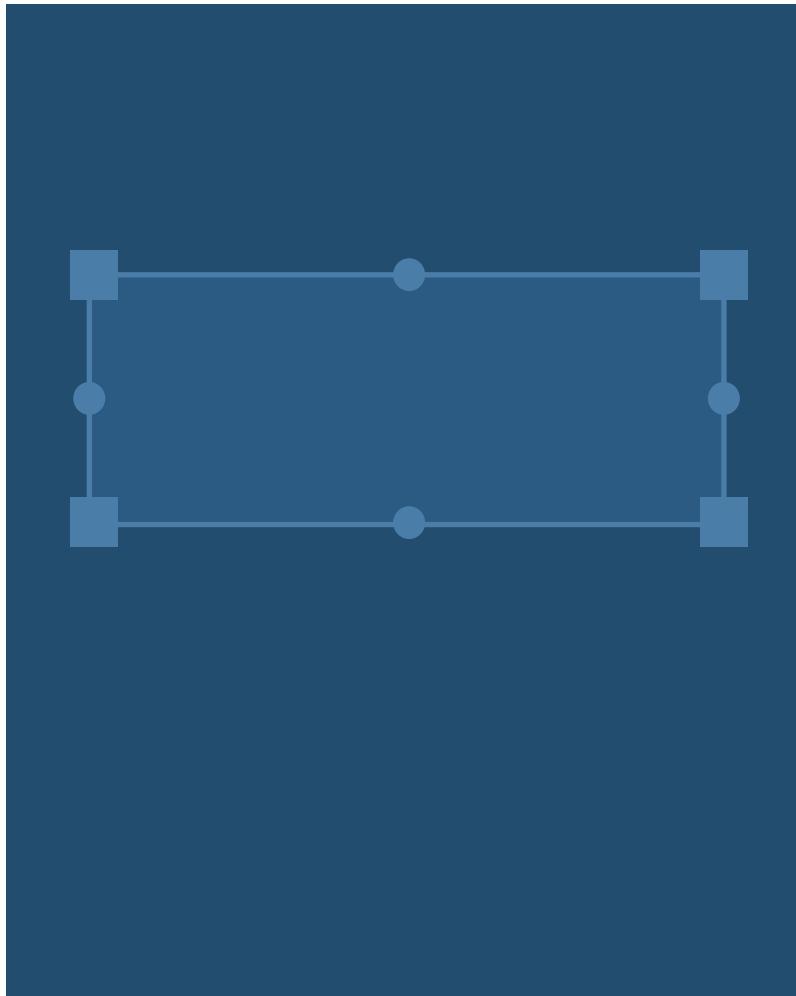
# ConstraintLayout Class

## Children leverage constraints

- Relative size/position
- Ratio-base size/position
- Group size/position distribution
  - Known as “chains”
- Weighted relationships
- Guideline-based size/position



# ConstraintLayout



**Should set horizontal & vertical constraints**

- Positions at 0,0 without constraints
- Can set more than one of each

**Setting constraints with the designer**

- Drag circle at mid-line to relationship

**Setting fixed size with the designer**

- Drag corner squares



# Creating the Activity UI

## Programmatically

- Use Java code to create class instances
- Relationships and properties set in code

## Layout files

- XML files describe View hierarchy
- Usually created using the Android Studio UI Designer



# Activity/Layout Relationship

**There is no implicit relationship**

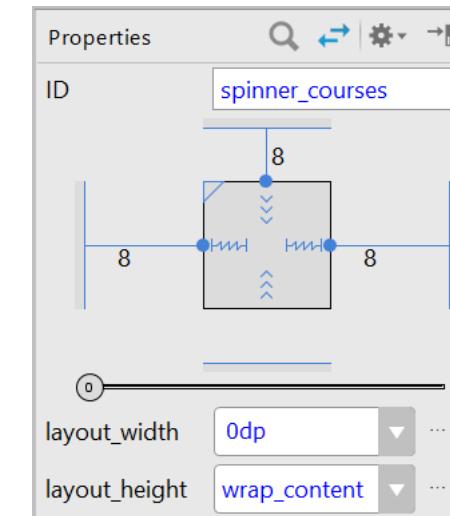
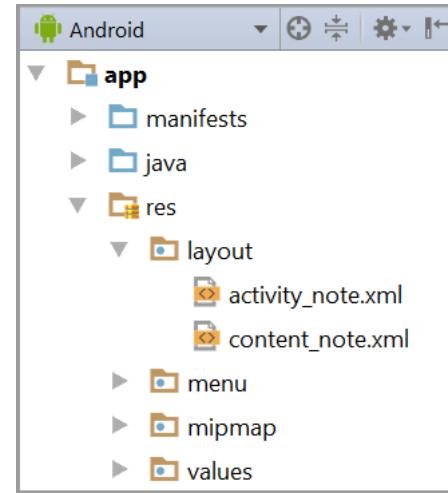
- Activity must load layout
  - Use setContentView
- Activity must request View reference
  - Use findViewById

**Relies on generated class R**

- Contains nested classes
- Layouts names in R.layout
- Id names in R.id

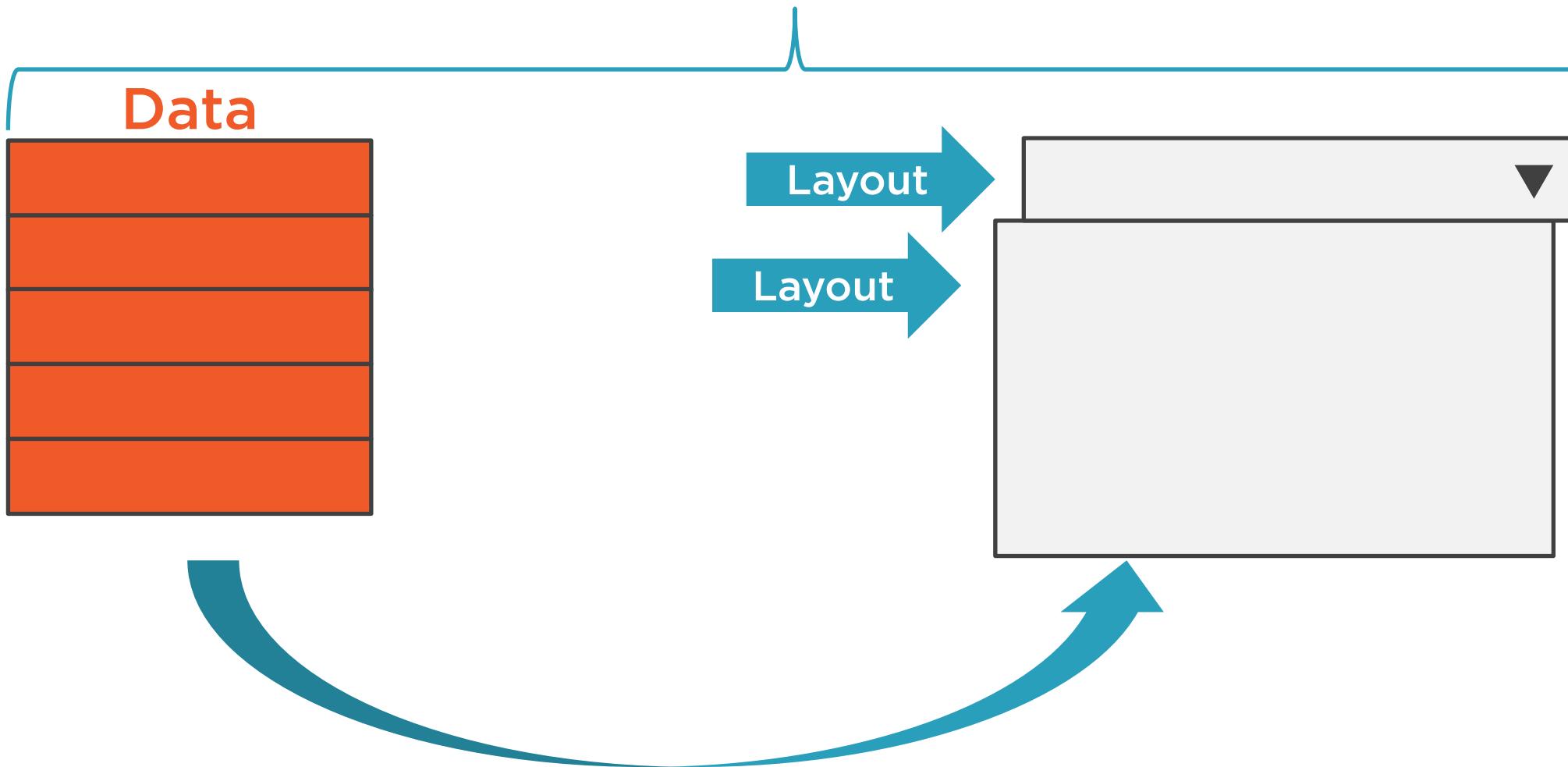


```
public final class R {  
    public static final class layout {  
        public static final  
            int activity_note = ... ;  
        // ...  
    }  
  
    public static final class id {  
        public static final  
            int spinner_courses = ... ;  
        // ...  
    }  
}
```



# Populating a Spinner

## Adapter



# Summary



## Activity/layout relationship

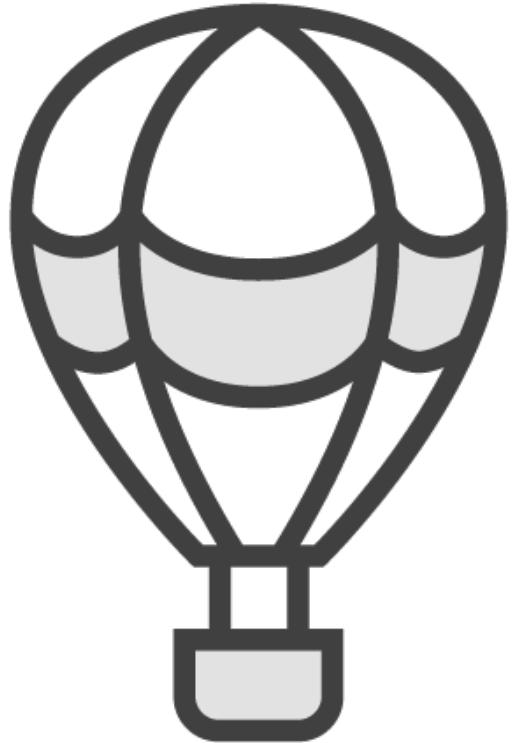
- No implicit relationship exists
- Must load layout
  - Use setContentView
- Must request layout View references
  - Use findViewById

## R class provides important constants

- Layout resources
  - R.layout
- View Id's
  - R.id



# Our App - The High-level View

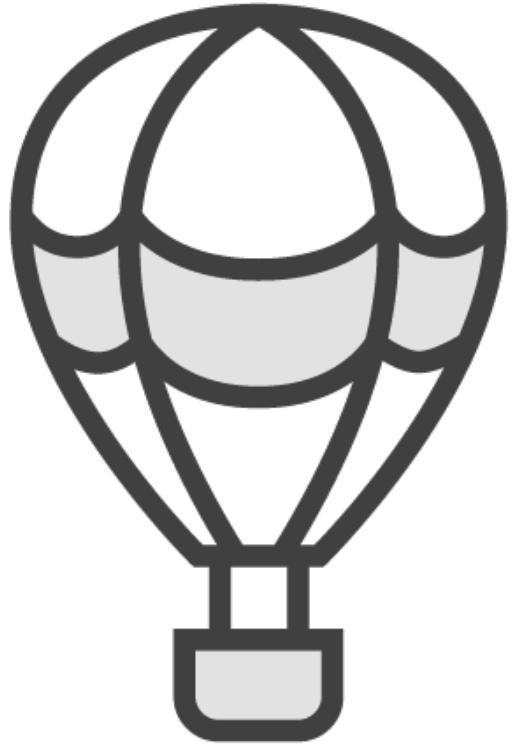


**We'll be building a note keeper app**

- Display list of notes
- Edit existing notes
- Create new notes



# Our App - The High-level View

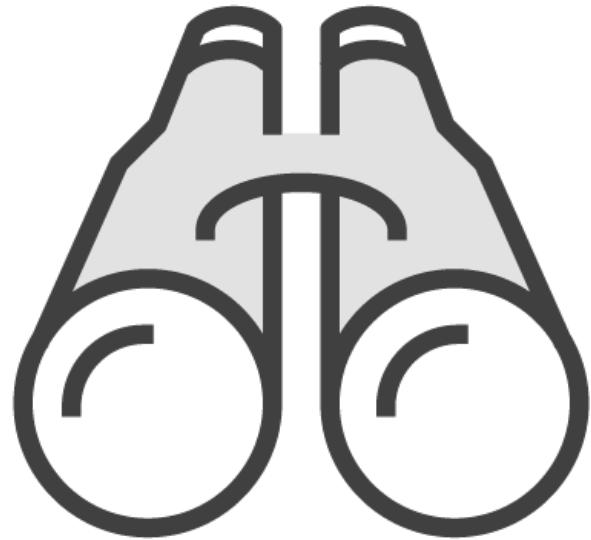


## Our app's definition of a “note”

- Course
  - Name of a course
  - Must correspond to an actual course
- Note title
  - User entered value to identify note
- Note text
  - User entered value that serves as the content of the note



# Our App - Some Long-term Plans



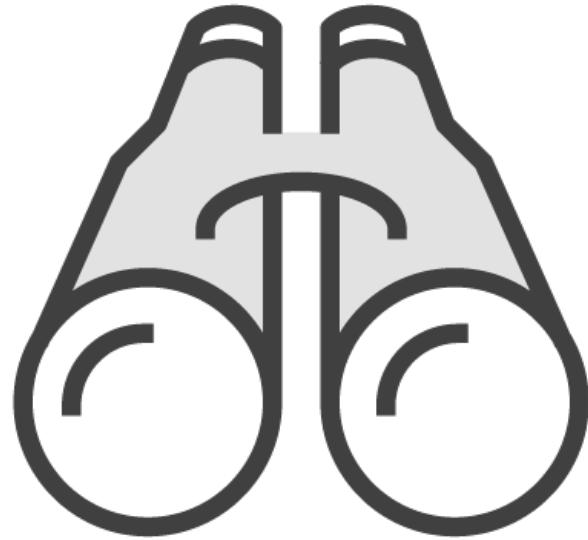
**Automated testing of logic and UI/UX**

**Card-style lists**

**Slide-out drawer navigation**



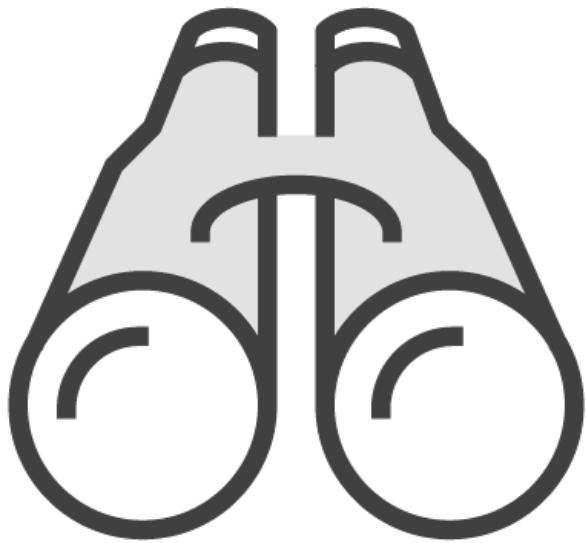
# Our App - Some Long-term Plans



- Track completed modules**
- Branding-based UI theme**
- Device and language adaptability**
- Support for users with accessibility needs**
- User customizable behaviors**



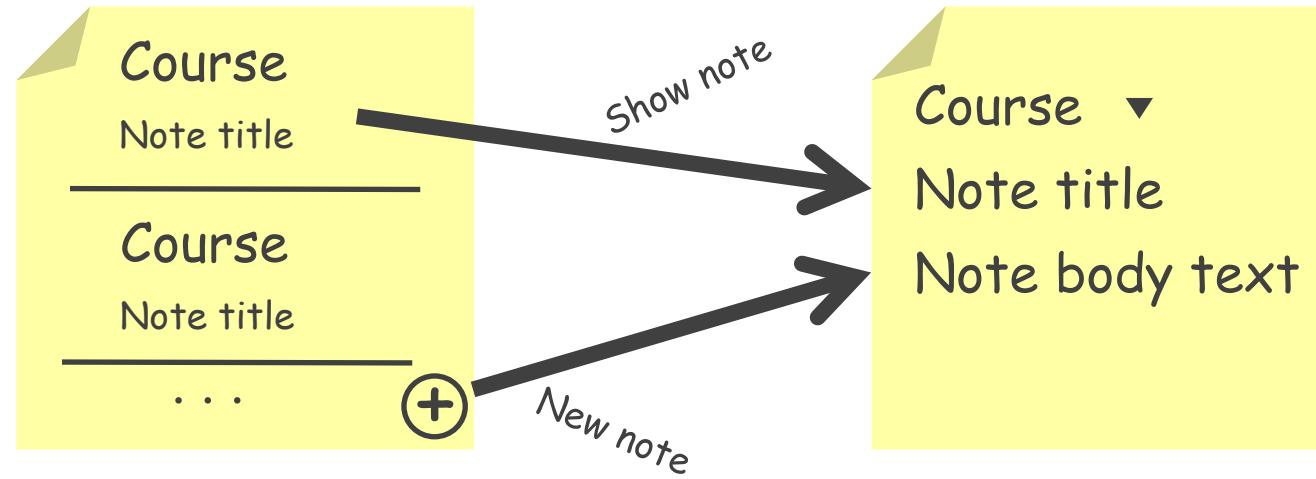
# Our App – Some Long-term Plans



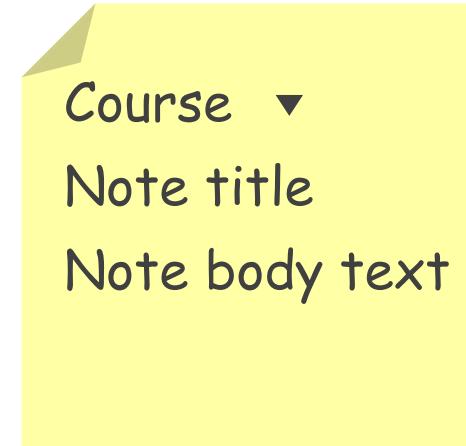
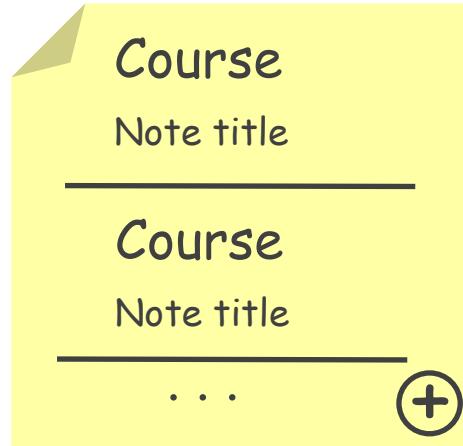
- Use SQLite to store and access our data**
- Make note data available to other apps**
- Display reminder notifications**
- Read/save data in the background**
- Display note information on home screen**



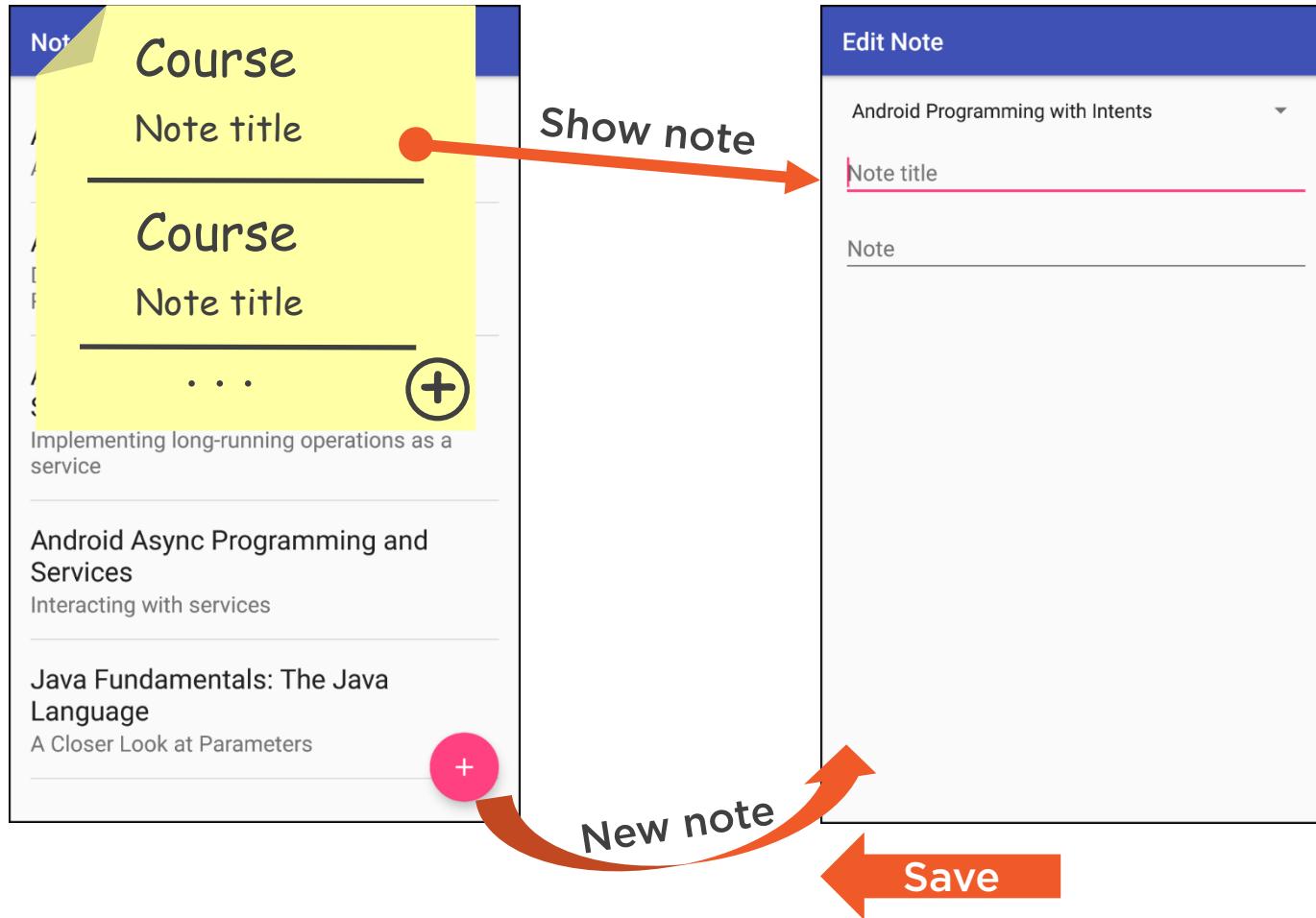
# Rough Design of Our App Starting Point



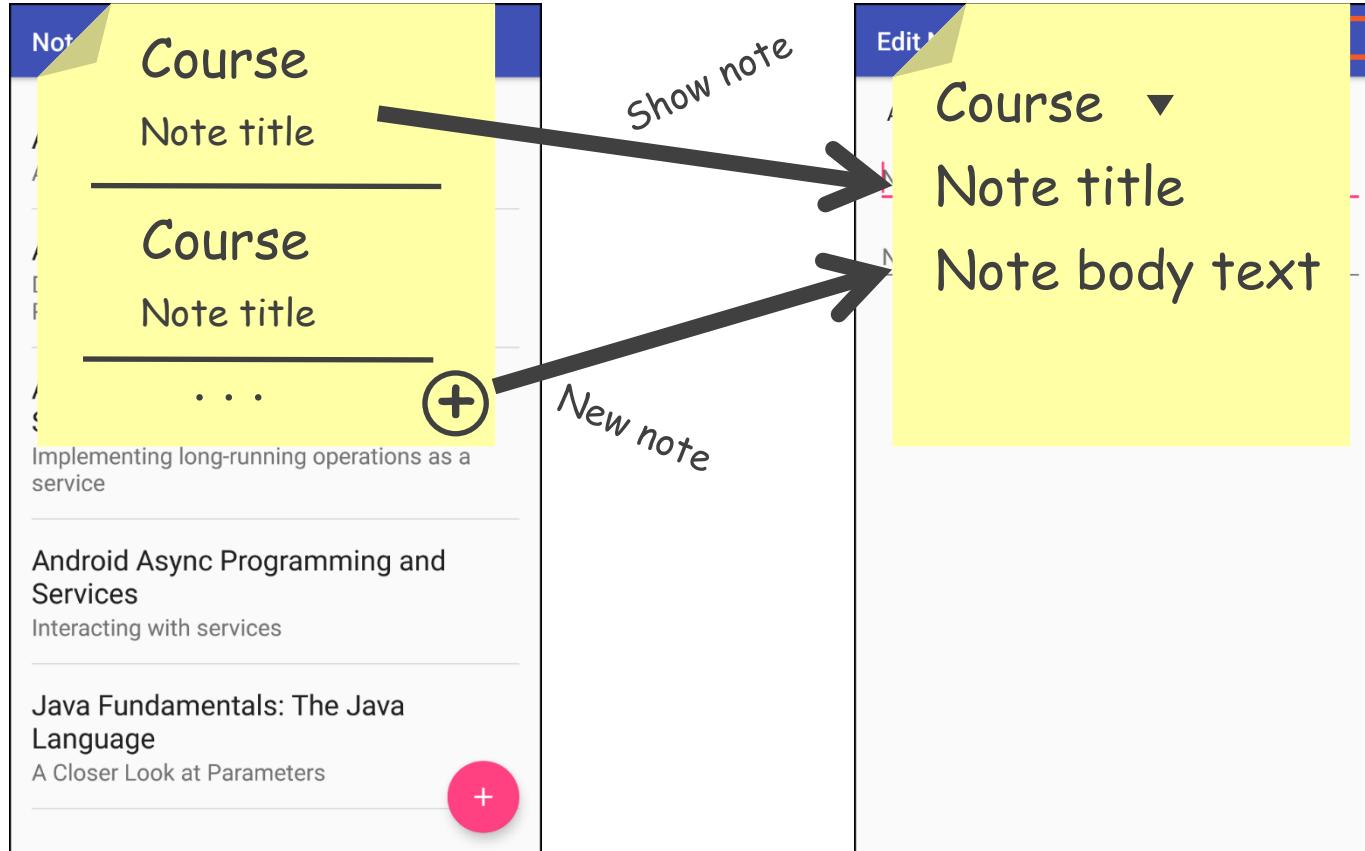
# Polished Design of Our App Starting Point

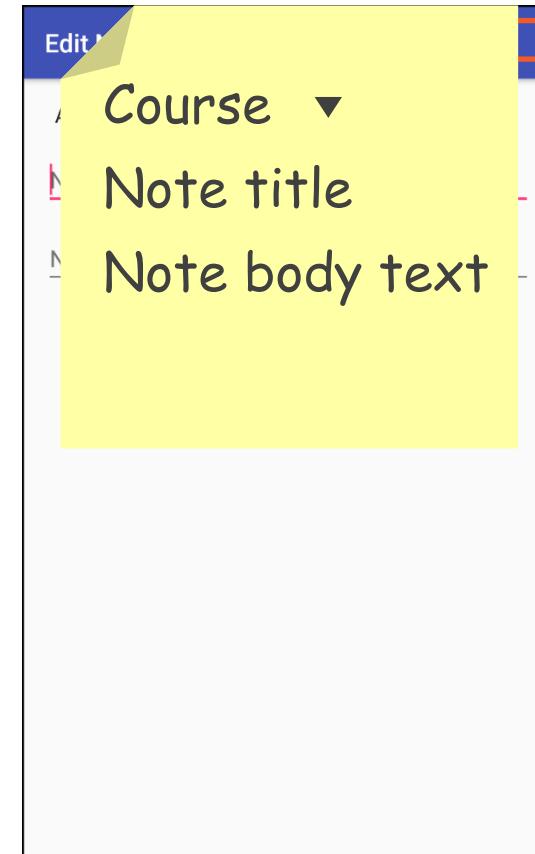
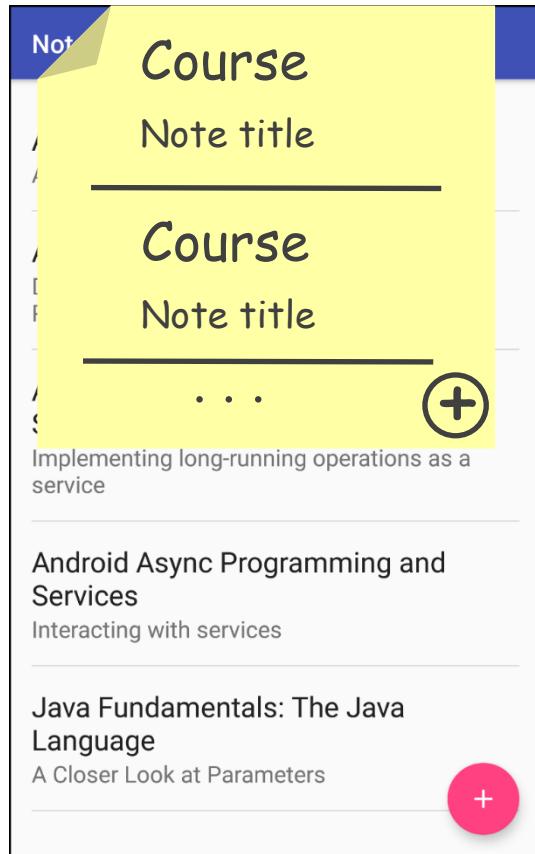


# Polished Design Our App Starting Point



# Designing Our App Starting Point





### NoteKeeper

Android Programming with Intents  
Android Late Binding and Intents

---

Android Programming with Intents  
Delegation and Callbacks through PendingIntents

---

Android Async Programming and Services  
Implementing long-running operations as a service

---

Android Async Programming and Services  
Interacting with services

---

Java Fundamentals: The Java Language  
A Closer Look at Parameters



### Edit Note

Android Programming with Intents

Note title

Note



# Summary



## Android Studio

- Complete Android dev environment

## Layout files

- Describe UI

## Activity

- Code and functionality to present UX

## Android Emulator

- Run and test apps from your desktop
- a.k.a. Android Virtual Device (AVD)



# What to Expect from This Course



**Building a Simple App**

**Designing and Planning Our App**

**Understanding Activity & Layout Interaction**

**Working with Activities and Activity Lifecycle**

**Using Option Menus and Digging Deeper into Activity Lifecycle**



# Goals of This Course Series

## Skills

Provide skills necessary  
to be an effective  
Android developer



## Certification

Provide the knowledge  
necessary to pass the  
Android Associate Developer  
exam



# What to Expect from This Course Series



**Understanding Android App Basics**

**Working with Android Tools & Testing**

**Enhancing the App Experience**

**Broadening App Appeal and Reach**

**Managing App Data with SQLite**

**Exposing Data & Info Outside Your App**

**Leveraging the Power of the Platform**



# This Is the Module Title in Titlecase

---



**Author Name**

AUTHOR TITLE

@authortwitter [www.authorsite.com](http://www.authorsite.com)



# Demo



**This bullet list is preset with animations**

**Use this layout to introduce your demo**

**How to do this one thing**

- Why we do it
- How we do it

**Then there's that thing**

**Don't forget to do this**

**We'll finish it off with this thing**





# Using the **Image Chunking** Slides

A slide template with a title bar and two image placeholders. Each placeholder contains a 'Click to add image' button and a small preview icon. Below each placeholder is a 'Click to add text' box.

Two Image Chunking

A slide template with a title bar and three image placeholders. Each placeholder contains a 'Click to add image' button and a small preview icon. Below each placeholder is a 'Click to add text' box.

Three Image Chunking

A slide template with a title bar and four image placeholders. Each placeholder contains a 'Click to add image' button and a small preview icon. Below each placeholder is a 'Click to add text' box.

Four Image Chunking

A slide template with a title bar and six image placeholders. Each placeholder contains a 'Click to add image' button and a small preview icon. Below each placeholder is a 'Click to add text' box.

Six Image Chunking

These layouts can be used as an alternative to a bulleted list.

They're built specifically for **photos** or **graphics** and look especially awesome when you incorporate icons from the **Pluralsight Icon Library**.

See them in action in the next 4 slides.



# Example of Image Chunking Two Items



**Jill Anderson**

Some information about this graphic goes here and four lines or fewer is best



**John Doe**

Some information about this graphic goes here and four lines or fewer is best



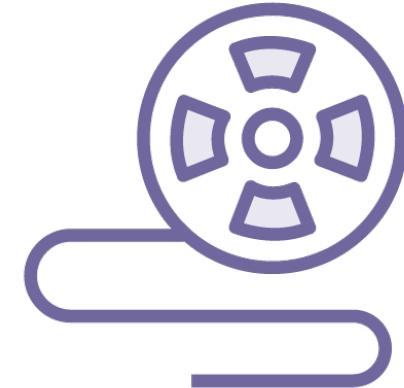
# Example of Image Chunking Three Items



**Clipboard**  
Some information  
goes here; three lines  
or fewer is best



**Book**  
Some information  
goes here; three lines  
or fewer is best



**Film**  
Some information  
goes here; three lines  
or fewer is best



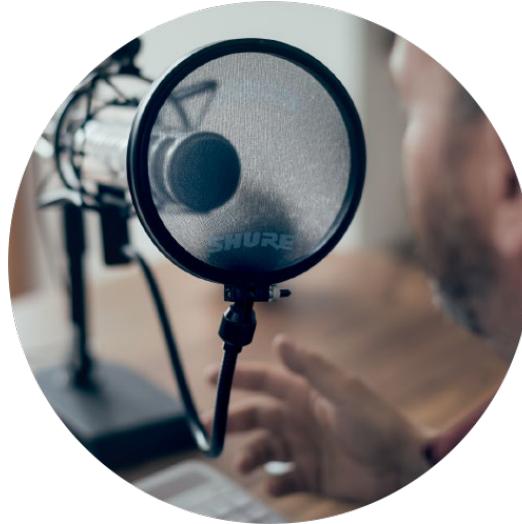
# Example of Image Chunking Four Items



Write



Create



Record



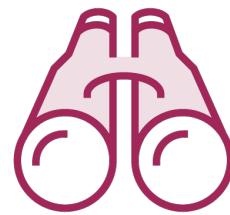
Learn



# Example of Image Chunking Six Items



Address book



Binoculars



Camera



Eyeglasses

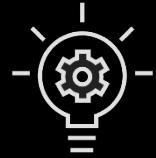


Megaphone



World

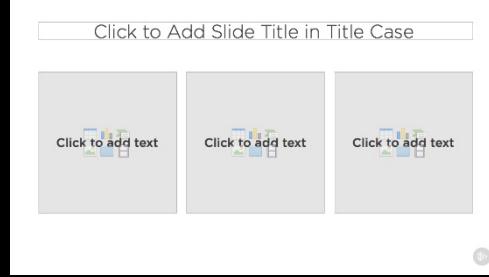




# Using the **Text Chunking Slides**



Two Text Chunking



Three Text Chunking

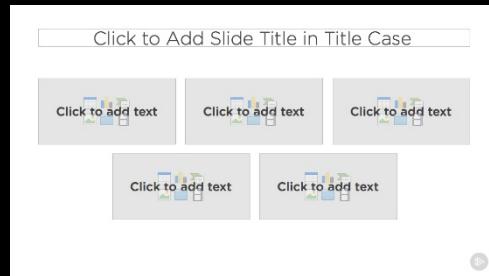
These layouts are intended to group chunks of text. Among other uses, they can be a great alternative to a bullet list.

Use **animations** to bring focus to the point you're speaking on one at a time, and/or use **color** to group points together.

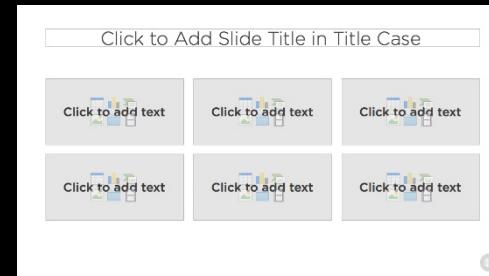
If you have more than six points to discuss, you may want to use a standard bullet list.



Four Text Chunking



Five Text Chunking



Six Text Chunking

We have provided some **example uses** of these layouts in the next few slides.



# Text Chunking **Two Items**

## Talking point one

Be concise and keep the text  
to four lines or fewer

## Talking point two

Be concise and keep the text  
to four lines or fewer



# Text Chunking Three Items

## Talking point one

Be concise and keep  
the text to four lines  
or fewer

## Talking point two

Be concise and keep  
the text to four lines  
or fewer

## Talking point three

Be concise and keep  
the text to four lines  
or fewer



# Text Chunking **Four Items**

This is the first talking point  
that should be kept to three  
lines or fewer

This is the second talking  
point that should be kept to  
three lines or fewer

This is the third talking point  
that should be kept to three  
lines or fewer

This is the fourth talking point  
that should be kept to three  
lines or fewer



# Text Chunking Five Items

## Talking point one

Keep the text to three  
lines or fewer

## Talking point two

Keep the text to  
three lines or fewer

## Talking point three

Keep the text to  
three lines or fewer

## Talking point four

Keep the text to  
three lines or fewer

## Talking point five

Keep the text to  
three lines or fewer



# Today's Mobile World

iPhone

Nexus 5

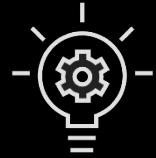
Lumia 950 XL

iPad

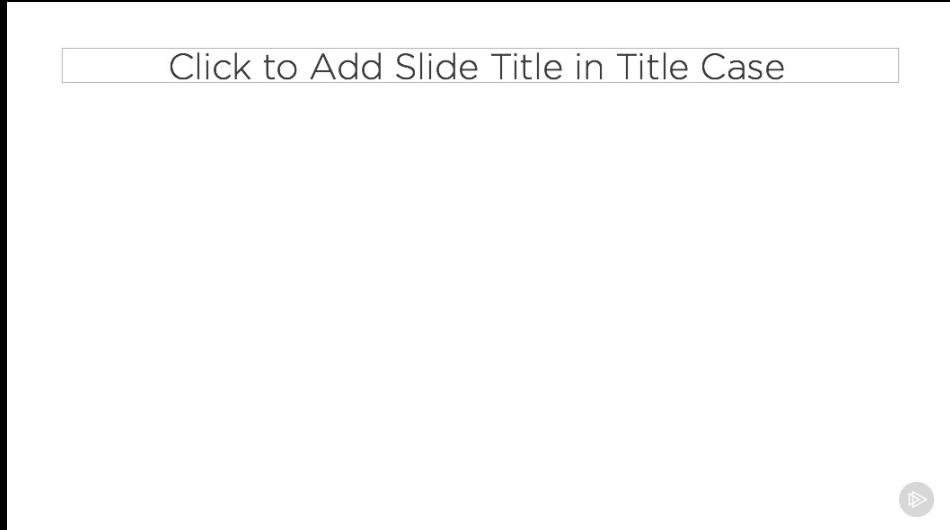
Nexus 7

Surface





# Using the **Title Only Slide**



Title Only

This is the slide you'll want to use when you just need a big space for a diagram, chart, or graphic.

Make sure you check out the training videos available on the **Author Kit** for design best practices.

If you need help bringing your ideas for this space to life, contact your Editor about getting help from one of our **Content Graphic Designers**. In most cases, you just need to submit a rough outline and let our designers work their magic. However, in some special cases, your Editor can get you in touch with a designer directly.

We included some possible starting points for you in the next few slides.



Remember, we are here to help!



# Customer Acquisition and Loyalty

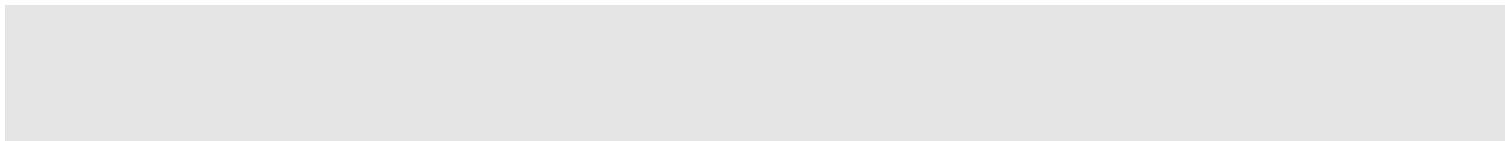
Observed higher sales



42%



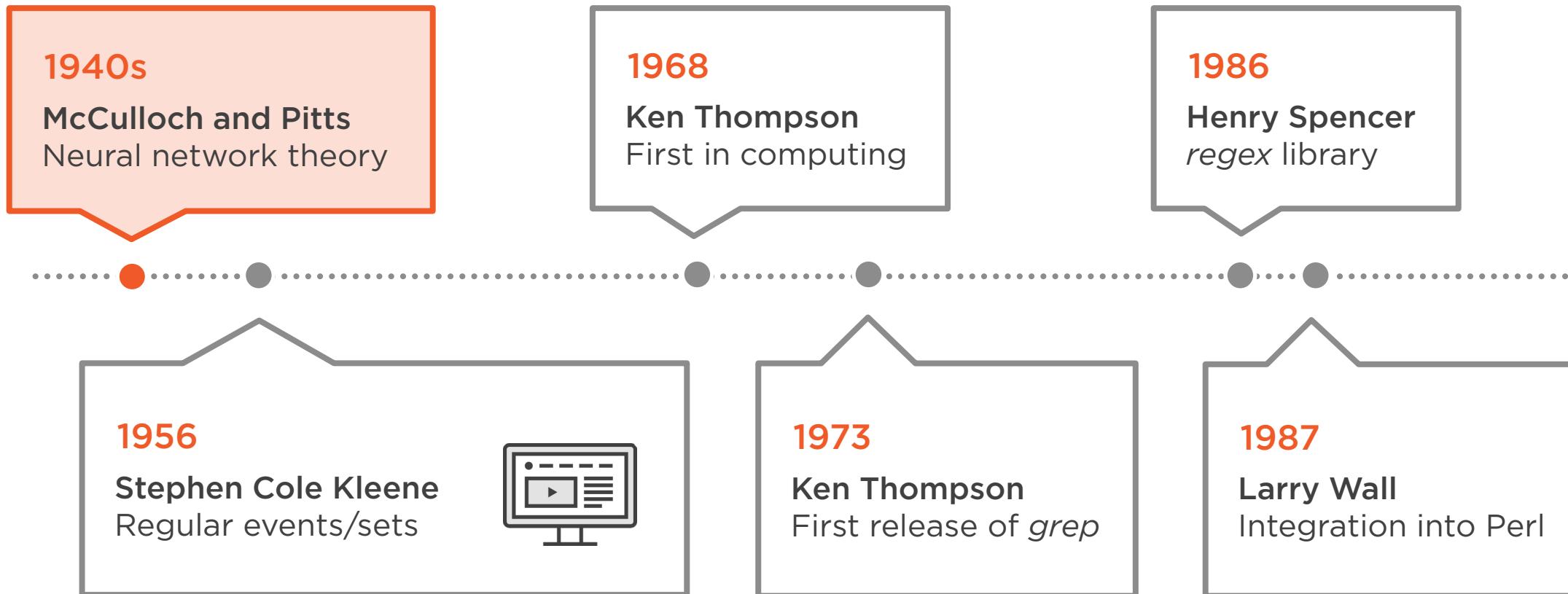
Observed more loyal customers



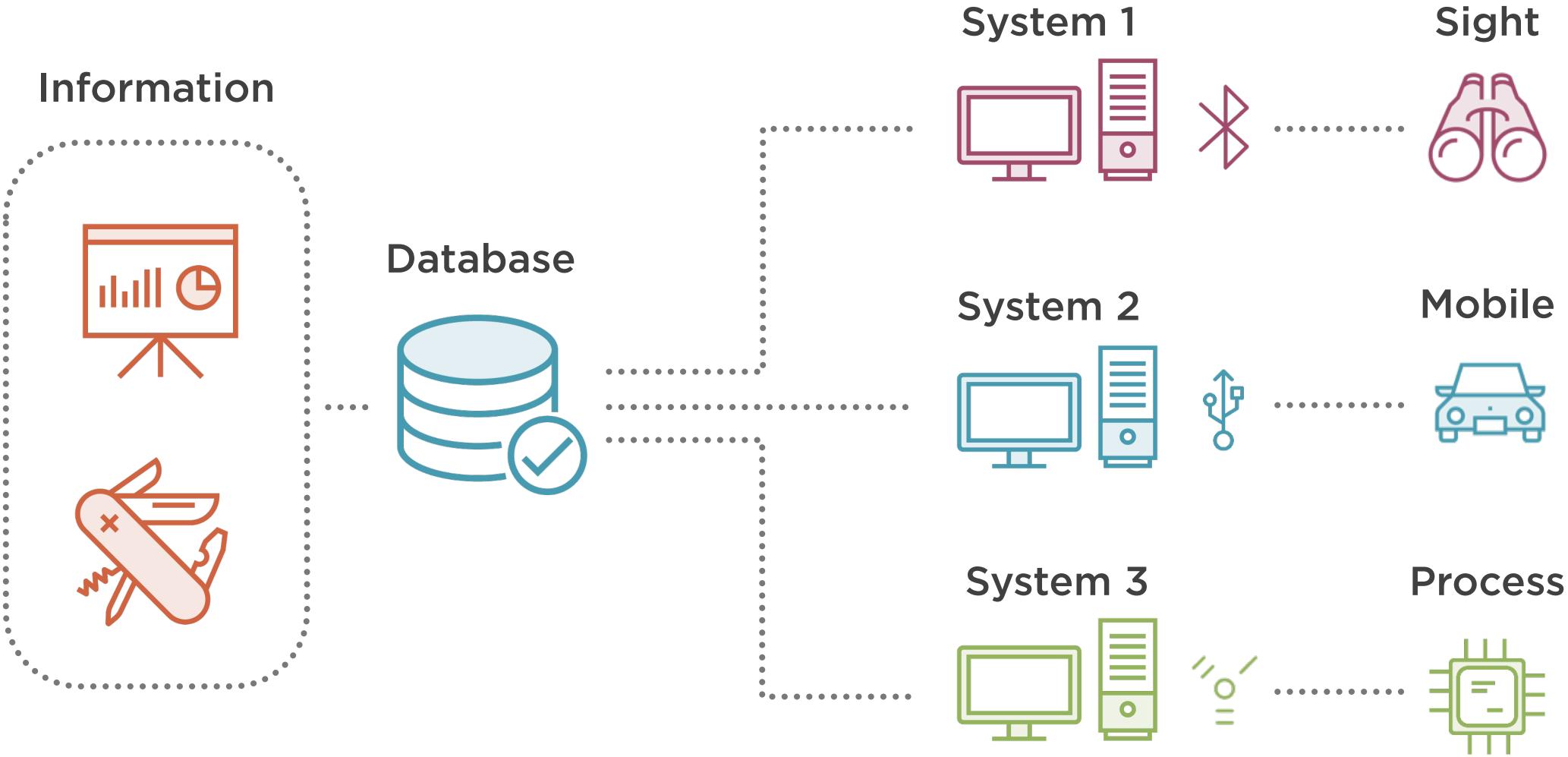
70%



# Timeline of Events



# Title Only Layout Example





# Using the **Code Slides**



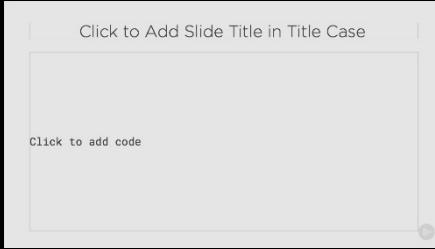
Code Top (Dark)



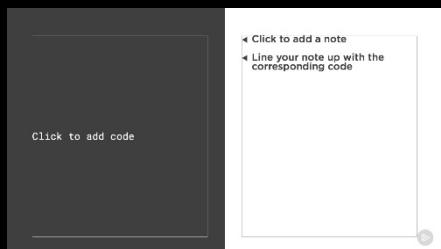
Code Top (Light)



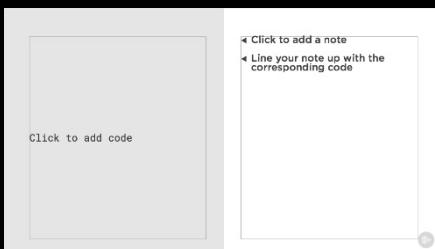
Code (Dark)



Code (Light)



Code Notes (Dark)



Code Notes (Light)

## Code Top Layouts

Use when you need a slide title and info about your code



Make use of the color palette to highlight code.

We recommend using the **Roboto Mono** typeface for your code slides. However, if you use a different font for code in your demos, feel free to use that instead to reinforce a consistent look.

## Code Layouts

Best for larger code snippets

## Code Left Layouts

Great for annotating code structure



```
<div class="row carousel-indicators">  
    <div style="background-color:red;" class="col-md-4" data-target="#homeCarousel" data-slide-to="0" class="active">  
<div class="row carousel-indicators">
```

Slide Title in Titlecase  
**Information about the code above**



```
<div class="row carousel-indicators">  
    <div style="background-color:red;" class="col-md-4" data-target="#homeCarousel" data-slide-to="0" class="active">  
<div class="row carousel-indicators">
```

---

Slide Title in Titlecase  
**Information about the code above**



# Code Snippet on Dark

```
<div class="row carousel-indicators">  
    <div style="background-color:red;" class="col-md-4" data-target="#homeCarousel" data-slide-to="0" class="active">  
        </div>  
    <div style="background-color:green;" class="col-md-4" data-target="#homeCarousel" data-slide-to="1">  
        </div>
```



# Code Snippet on Light

```
<div class="row carousel-indicators">  
    <div style="background-color:red;" class="col-md-4" data-target="#homeCarousel" data-slide-to="0" class="active">  
        </div>  
    <div style="background-color:green;" class="col-md-4" data-target="#homeCarousel" data-slide-to="1">  
        </div>
```



Put code on this side

```
var proto = {  
  foo: 'Hello World'  
};
```

```
function Bar(){}  
Bar.prototype = proto;
```

```
var baz = new Bar();
```

```
console.log(baz.foo);
```

◀ Line up with these notes

◀ Set up prototype object

◀ Constructor function  
and set prototype property

◀ Create instance

◀ Call inherited member



Put code on this side

```
var proto = {  
  foo:'Hello World'  
};
```

```
function Bar(){}  
Bar.prototype = proto;
```

```
var baz = new Bar();
```

```
console.log(baz.foo);
```

◀ Line up with these notes

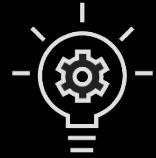
◀ Set up prototype object

◀ Constructor function  
and set prototype property

◀ Create instance

◀ Call inherited member



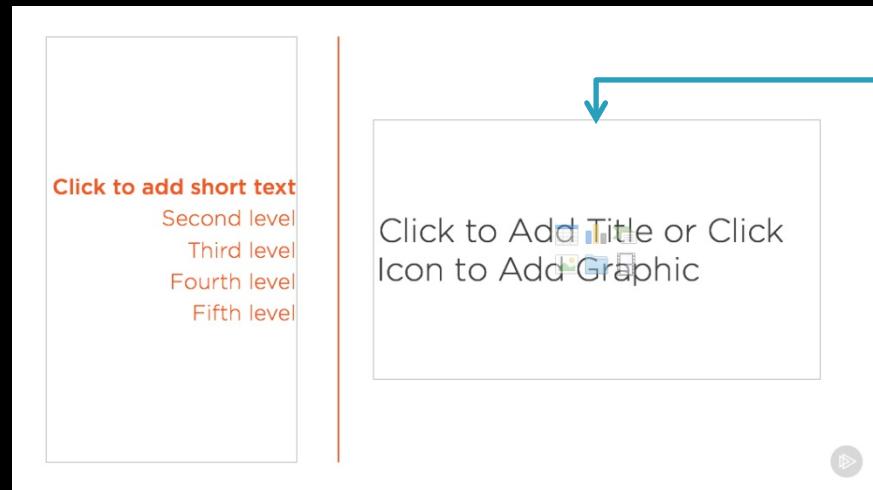


# Using Bullet List Slides

We've provided some bullet list layouts to accommodate various quantities of information.

## Content left | Title/Image right

Intended for bullet text that is shorter and titles/images that are larger

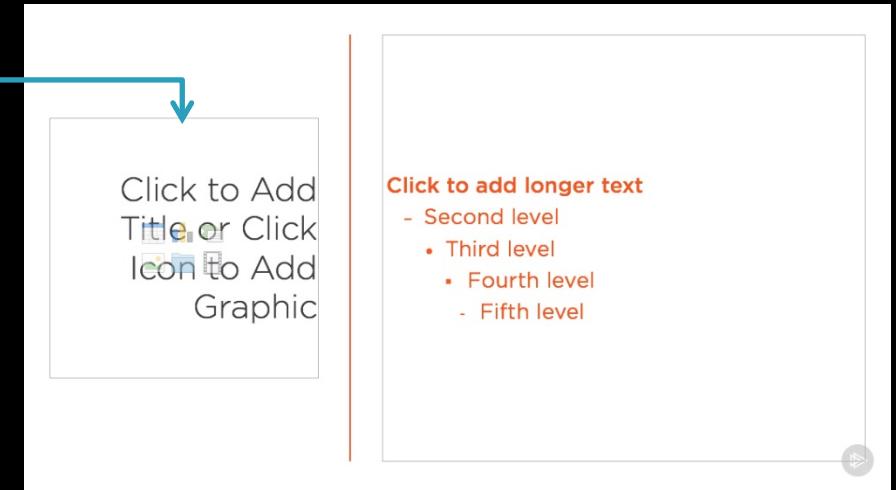


This diagram shows a slide layout with two main sections. On the left, there is a large white box containing placeholder text: "Click to add short text" followed by a nested list from "Second level" to "Fifth level". On the right, there is a smaller white box with the placeholder "Click to Add Title or Click Icon to Add Graphic". A blue arrow points from the text in the first box down to the second box, indicating they are part of the same slide. A small circular navigation icon is at the bottom right.

Content | Image/Title

## Title/Image left | Content right

Intended for bullet text that is longer and titles/images that are smaller



This diagram shows a slide layout with two main sections. On the left, there is a large white box containing placeholder text: "Click to Add Title or Click Icon to Add Graphic" followed by a nested list from "Second level" to "Fifth level". On the right, there is a smaller white box with the placeholder "Click to add longer text". A blue arrow points from the text in the first box down to the second box, indicating they are part of the same slide. A small circular navigation icon is at the bottom right.

Image/Title | Content



**Animation built in**

**Bullet alternative**

**Sentence fragments**

**List of things**

**Procedure list**

**Talking points**

Title or Relevant Graphic



**Animation built in**

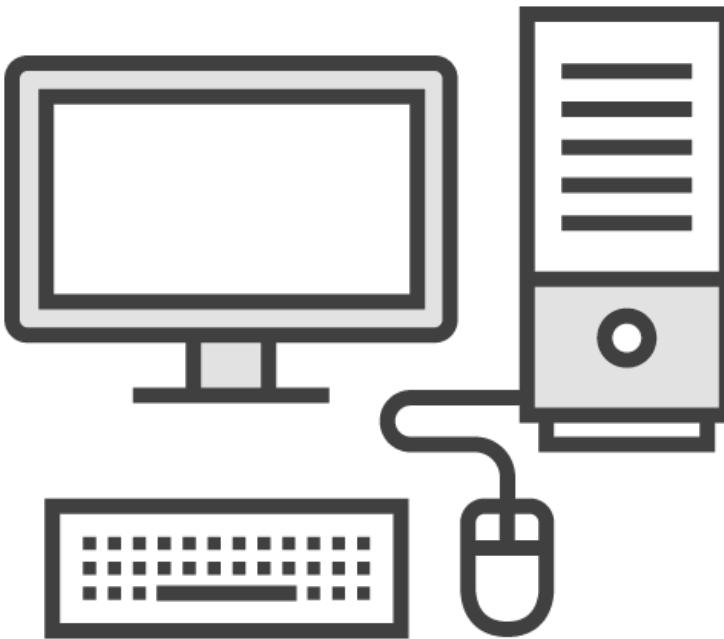
**Bullet alternative**

**Sentence fragments**

**List of things**

**Procedure list**

**Talking points**



# Title or Relevant Graphic

**Animation built in**

**Bullet alternative**

**Room for a bit more text**

**Use this layout for**

- Longer sentence fragments
- List of things
- Procedure list
- Talking points





**Animation built in**

**Bullet alternative**

**Room for a bit more text**

**Use this layout for**

- Longer sentence fragments
- List of things
- Procedure list
- Talking points



# Title Space with Image



**Animation built in**

**Bullet alternative**

**Room for a bit more text**

**Use this layout for**

- Longer sentence fragments
- List of things
- Procedure list
- Talking points





**Graphic on left should fill the entire space**

- Graphic must be high quality and royalty free

**Graphic and text animation is built in**





# Comparison Slide

Use this slide if you need to compare single items or groups of items.

Click to Add Slide Title in Title Case

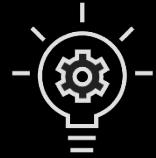
Compare item one	Compare item two
Click to add text	Click to add text



# Comparison Example

Functional group	Objectives
Configure and administer security	Manage vSphere storage virtualization
Configure advanced networking	Configure software-defined storage
Configure advanced storage	Configure vSphere storage multipathing and failover
Administer and manage resources	Perform advanced VMFS and NFS configurations and upgrades
Configure availability solution	
Deploy and consolidate vSphere	

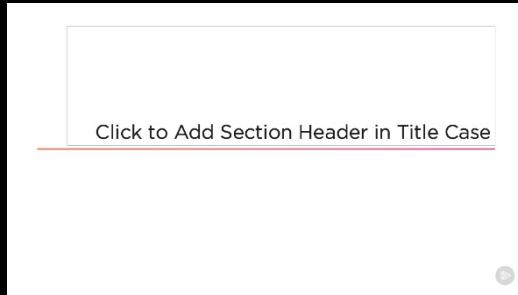




# Other Slides

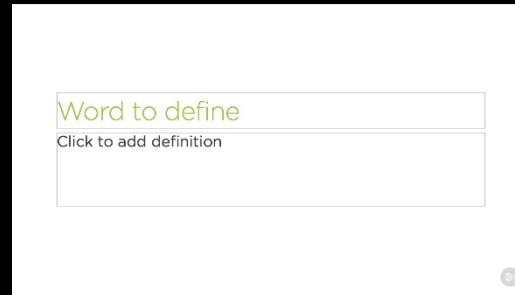
The following self-explanatory slides are a good way of adding diversity into the flow of your course.

Use them purposefully.



A slide template with a white background. At the top left is a large, empty rectangular box. Below it, a horizontal line with red text above it reads "Click to Add Section Header in Title Case". A small circular arrow icon is at the bottom right.

Section Header



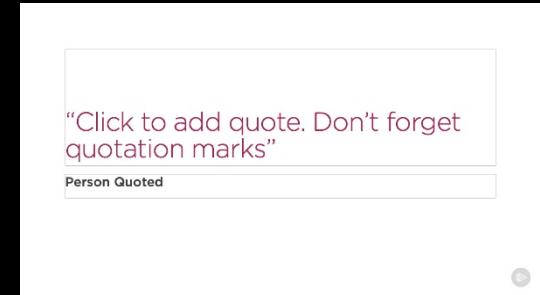
A slide template with a white background. At the top left is a green rectangular box containing the text "Word to define". Below it is a smaller white rectangular box with the text "Click to add definition". A small circular arrow icon is at the bottom right.

Definition



A slide template with a teal background. In the center is a white rectangular box containing the text "This is a short, important statement to bring attention to something." A small circular arrow icon is at the bottom right.

Important Statement



A slide template with a white background. At the top left is a pink rectangular box containing the text "Click to add quote. Don't forget quotation marks". Below it is a smaller white rectangular box with the text "Person Quoted". A small circular arrow icon is at the bottom right.

Quotation



# Section Heading

---



# Word Definition

Here is where you put the definition. This is one of the few places where complete sentences are appropriate. Be sure to cite your source.



This is a short, important statement to bring attention to something.



“Using quotes in your slides can be powerful if used sparingly.”

**Heather Ackmann**

