



Android lecture 3

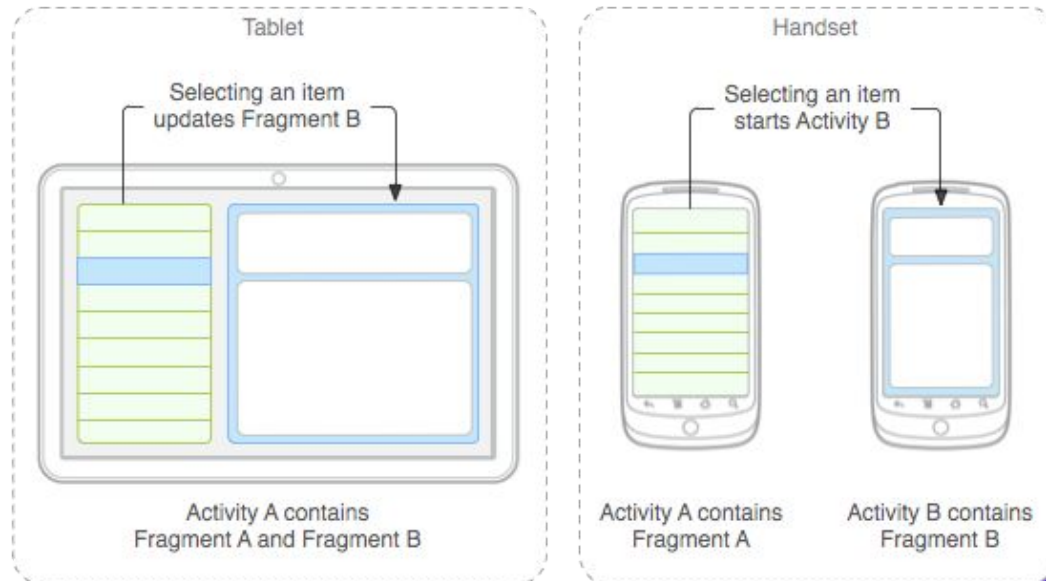
Fragments, Notifications, Data persistence

Agenda

- Fragments
 - Inflating
 - Lifecycle
- Adapters
- Notifications
- Data persistence
 - File
 - SharedPreferences
 - Database
 - Content provider

Fragment

- Simplify create UI for phones and tablets
- ~~android.app.Fragment~~ - Added API 11, deprecated API 28
- ~~android.support.v4.app.Fragment~~ - replaced by jetpack
- androidx.fragment.app.Fragment - jetpack



Fragment - add statically

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/fragment_id"
        android:tag="some_string"
        android:name="packagename.class"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

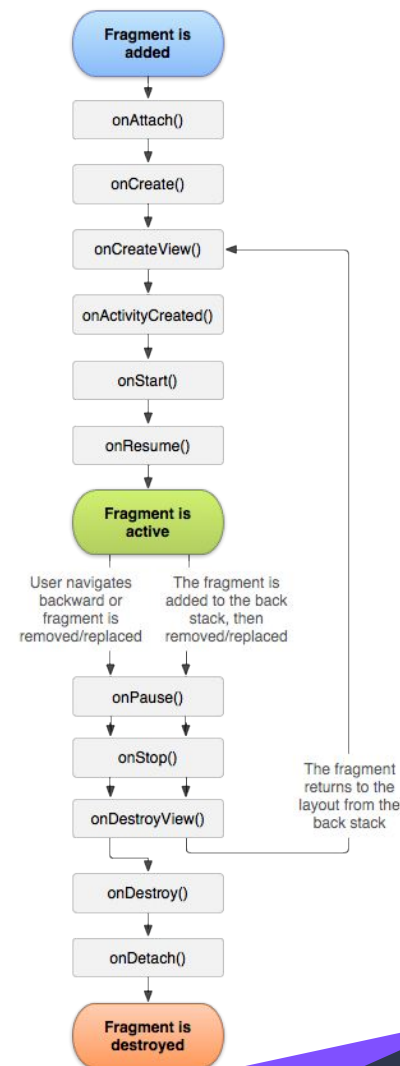
Fragment - Activity.kt

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_user)  
  
    supportFragmentManager.beginTransaction()  
        .add(R.id.fragment_container, MyFragment.newInstance())  
        .addToBackStack(null)  
        .commit()  
}
```

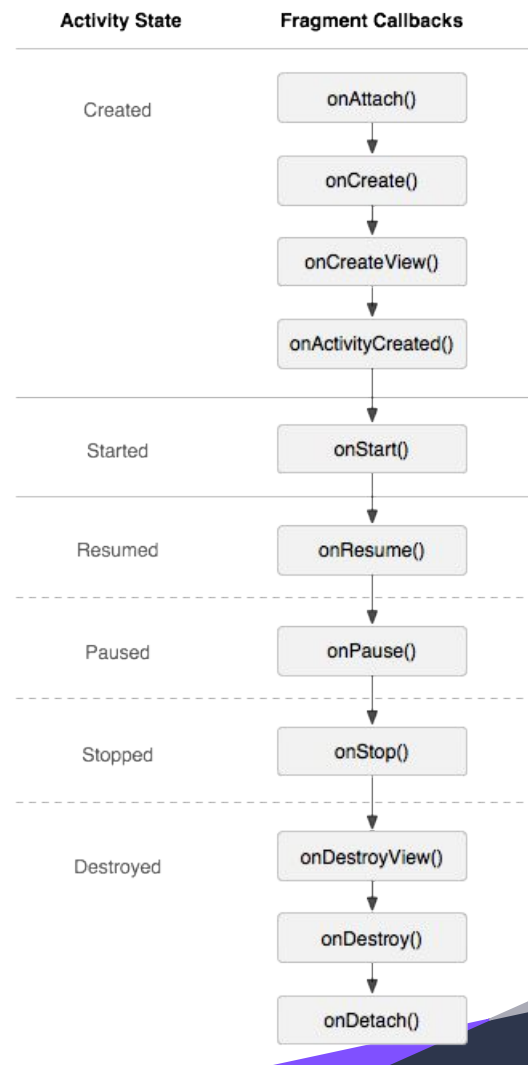
Fragment - states

- Is in same state as host activity
- Resumed
 - Fragment is visible in the running activity
- Paused
 - Another activity in foreground, but hosting activity is still visible
- Stopped
 - Fragment is not visible
 - Hosting activity is stopped or fragment is removed from the activity, but added to backstack.
 - Alive, can be killed by hosting activity

Fragment - lifecycle



Fragment inside Activity lifecycle



Fragment - lifecycle

- `onAttach(Activity)`
 - Fragment is associated with the activity
 - Set activity as a listener
- `onCreate(Bundle)`
 - Initial creation of fragment
 - Process fragment extras
 - Not called when fragment is retained across Activity re-creation
- `onCreateView(LayoutInflater, ViewGroup, Bundle)`
 - Called when view hierarchy needs to be created

Fragment - lifecycle

- `onActivityCreated(Bundle)`
 - Activity `onCreate()` completed
 - Get references to views
- `onViewStateRestored(Bundle)`
 - All saved state of the view hierarchy was restored
- `onStart()`
 - Fragment visible to user (same as `Activity.onStart()`)
- `onResume()`
 - Fragment interact with user (based on hosting container)
 - Same as `Activity.onResume()`

Fragment - lifecycle

- onPause()
 - Not interact with user anymore
 - Paused activity or fragment manipulation
- onStop()
 - No longer visible
 - Stopped activity or fragment manipulation
- onDestroyView()
 - Disconnect fragment from view hierarchy created in onCreateView()
- onDestroy
 - Fragment going to be destroyed
 - Cleanup all resources
 - Not called for retained fragments
- onDetach
 - Detach fragment from activity
 - Remove activity listeners

Retained fragment

- Call `Fragment#setRetainInstance(true)`
- Survive configuration change
- Views needs to be recreated
- `Fragment#onCreate()` is not called for retained instances
- Usually for background work or data caching

Headless fragment

- Fragment without UI
- Often retained fragment
- `Fragment#onCreateView()` returns null

Fragment and Activity

- Fragment is not working without activity
- Activity can call fragment methods directly
- Fragment defines interface to be implemented by Activity to handle fragment requirements

Fragment - passing data

```
class DemoFragment: Fragment() {  
  
    companion object {  
        @JvmStatic  
        fun newInstance(username: String): DemoFragment {  
            val fragment = DemoFragment()  
            fragment.arguments = bundleOf(  
                "Username" to username,  
                "id" to 1001  
            )  
  
            return fragment  
        }  
    }  
}
```

- Android calls non-params constructor when restoring fragments
- Constructor with parameters will not be called

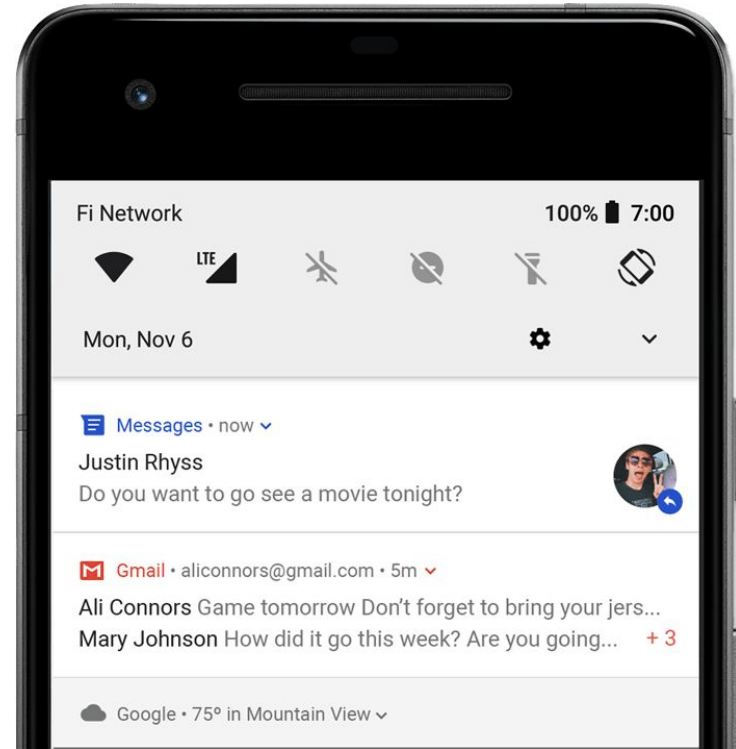
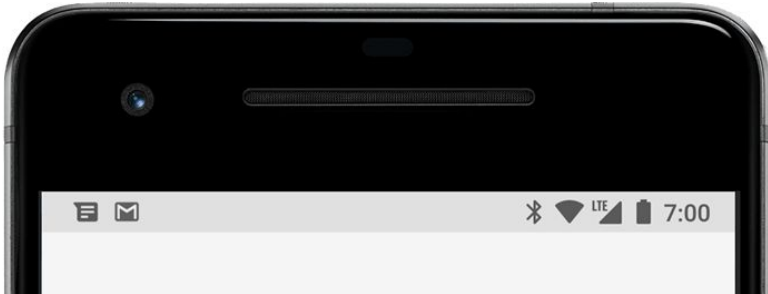
Exercise

1. Inflate LoginFragment in LoginActivity statically
2. Inflate UserFragment in UserActivity dynamically
3. Fill UserFragment data

Notifications

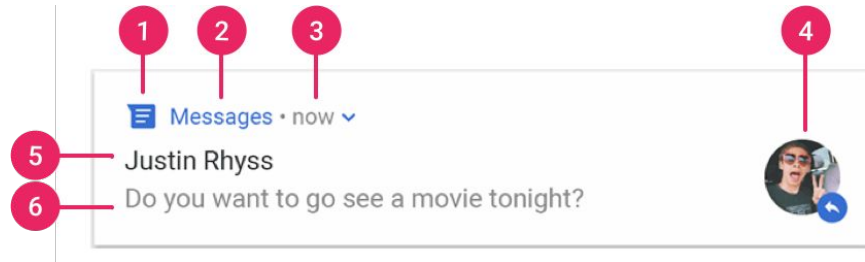
Notifications

- Notify user, when not use your app
- Mandatory
 - Small icon
 - Content title
 - Content text
 - Notification channel - Android 8.0+



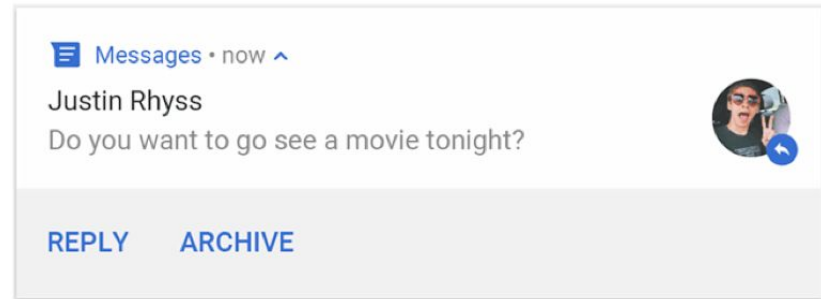
Notification

1. Small icon - mandatory parameter
2. App name - provided by system
3. Time stamp: provided by system,
 - override by `setWhen()`
 - Hide `setShowWhen(false)`
4. Large icon - optional
5. Title - optional
6. Text - optional



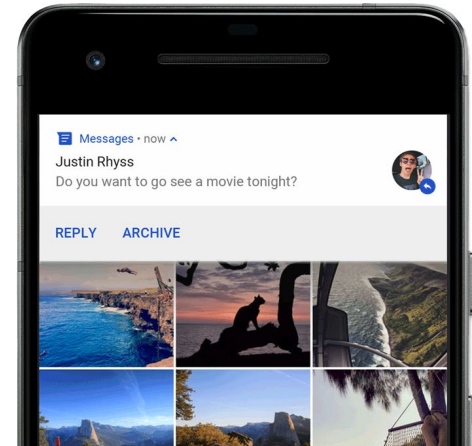
Notification actions

- Quick actions
- Inline reply action Android 7.0+



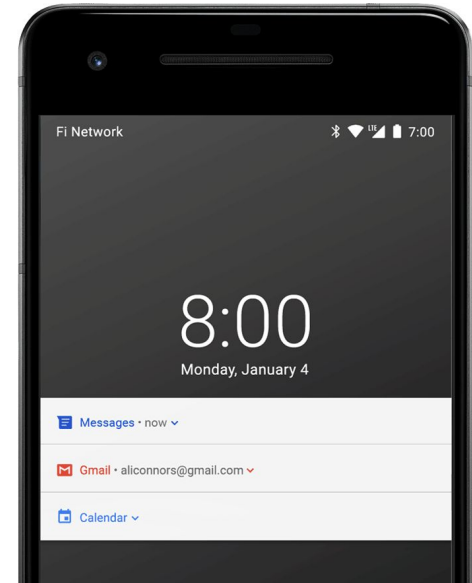
Heads-up notifications

- Heads-up notifications
 - Small floating window
 - Shows action buttons to handle action without leaving app
 - Only for high priority notifications
 - If the user's activity is in full screen mode (app uses `fullScreenIntent`)
 - Notification has high priority and uses ringtones or vibrations
- Android 5.0+



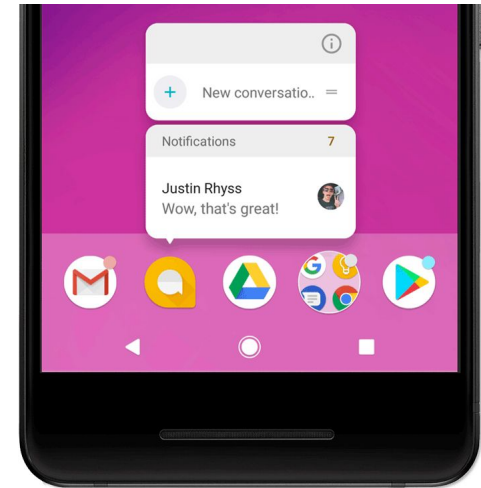
Lock screen notifications

- Possibility to set which informations when device is locked
 - VISIBILITY_PUBLIC - Shows full content
 - VISIBILITY_SECRET - Do not show at all
 - VISIBILITY_PRIVATE - Show icon and title, hide content
- Android 5.0+



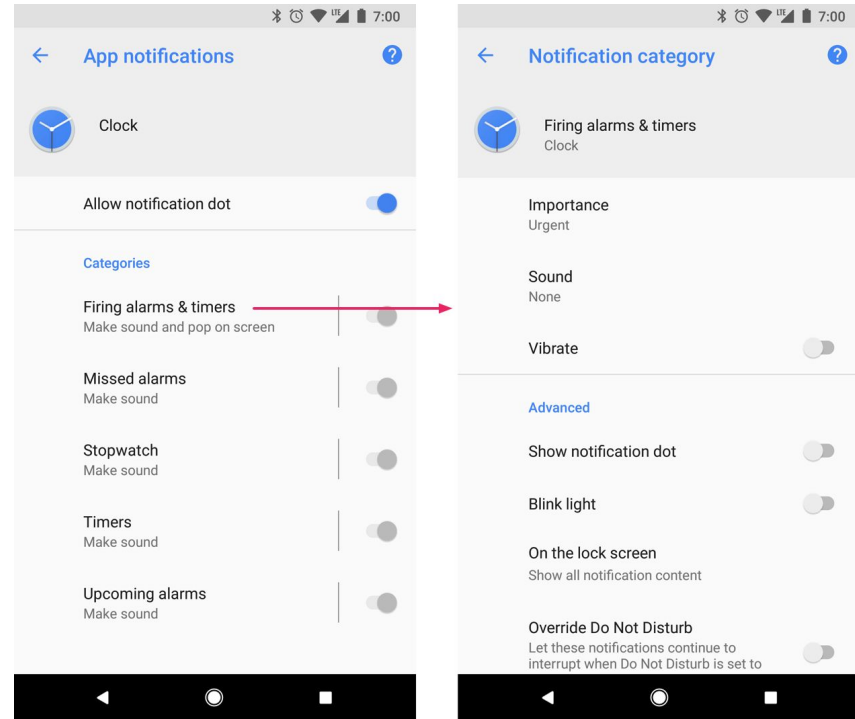
Notifications dots

- Dots on launcher app icons
- Shows notification content on long click
- Android 8.0+



Notification channels

- Notification categories
- User can manage notifications in single category
- Override DnD
- System show number of deleted channels
- Android 8.0+



Key classes

- android.app.Notification
- ~~android.support.v4.app.NotificationCompat~~
 - ~~Action~~
 - ~~Builder~~
 - ~~*Style~~
- androidx.core.app.NotificationCompat
 - Action
 - Builder
 - *Style
- android.app.NotificationManager
- ~~android.support.v4.app.NotificationManagerCompat~~
- androidx.core.app.NotificationManagerCompat
- android.app.NotificationChannel

Notification

- Use `NotificationCompat.Builder`
 - Handles compatibility
- `NotificationManagerCompat.notify(int id, Notification)`
- Possible to set pending intent for actions
- Priority affects position in drawer
- Developer responsibility to handle navigation when user opens application from notification

Create notification

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Much longer text that cannot fit one line...")
    .setStyle(NotificationCompat.BigTextStyle()
        .bigText("Much longer text that cannot fit one line..."))
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
```

Create channel

```
private fun createNotificationChannel() {  
    // Create the NotificationChannel, but only on API 26+ because  
    // the NotificationChannel class is new and not in the support library  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        val name = getString(R.string.channel_name)  
        val descriptionText = getString(R.string.channel_description)  
        val importance = NotificationManager.IMPORTANCE_DEFAULT  
        val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {  
            description = descriptionText  
        }  
        // Register the channel within the system  
        val notificationManager: NotificationManager =  
            getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager  
        notificationManager.createNotificationChannel(channel)  
    }  
}
```

Setup notification action

```
// Create an explicit intent for an Activity in your app
val intent = Intent(this, AlertDetails::class.java).apply {
    flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
}
// Create the TaskStackBuilder
val resultPendingIntent: PendingIntent? = TaskStackBuilder.create(this).run {
    // Add the intent, which inflates the back stack
    addNextIntentWithParentStack(intent)
    // Get the PendingIntent containing the entire back stack
    getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT)
}
val builder = NotificationCompat.Builder(this, CHANNEL_ID)
    ....
    // Set the intent that will fire when the user taps the notification
    .setContentIntent(resultPendingIntent)
    .setAutoCancel(true)
```

Fire notification

```
with(NotificationManagerCompat.from(this)) {  
    // notificationId is a unique int for each notification that you must define  
    notify(notificationId, mBuilder.build())  
}
```

Exercise

4. Fire notification when user tap on search button
5. Add notification action to open UserActivity

Persistence

Persisting data - files

- Standard Java API for file operations

Internal storage

- Always available
- For private data
- Removed with application uninstall
 - <https://medium.com/inloopx/samsung-tablets-are-not-removing-application-files-after-uninstall-45cc22ace56a>
- Cache

Internal storage

- `Context.getFilesDir()`
 - File representing internal directory for your app
- `Context.openFileOutput(filename: String, mode: Int)`
 - Filename - name of file
 - Mode - specify access to file
 - `MODE_PRIVATE` - accessible by apps with same UID
 - `MODE_APPEND` - append data instead of erasing file
 - `MODE_WORLD_READABLE` - Deprecated API 17, SecurityException API 24
 - `MODE_WORLD_WRITEABLE` - Deprecated API 17, SecurityException API 24
- `Context.openFileInput(filename: String)`
 - Filename - name of file

Internal storage - cache

- `Context.getCacheDir()`
 - File representing internal directory for app temporary files
 - System can delete these files, when is running low on storage
 - 3rd party cleaner apps often clear cache
 - Delete these files when are not longer needed
 - Presence of these files should not affect your application
 - It can just slow down app, need to download some resources

Internal storage - sharing data

- Data can be shared via `FileProvider`
 - Allows to specify shared directories
 - Implicit intent to pick specific files

External storage

- External storage != SD Card
- Not always available
- World readable
- Uninstall remove files in `Context.getExternalFilesDir()`
- Lot of API changes between android versions
- Often modified by vendors

External storage

- Requires permissions
 - `android.permission.WRITE_EXTERNAL_STORAGE`
 - `android.permission.READ_EXTERNAL_STORAGE`
 - Since API 19 permissions are not needed for private files
- Developer responsibility to check if the external storage is available
- Public files
 - Available to the other apps and user
 - Downloaded files
- Private files
 - Files to be deleted with app uninstall
 - Accessible to other, but no value for them
 - Temp downloaded files, ringtones,

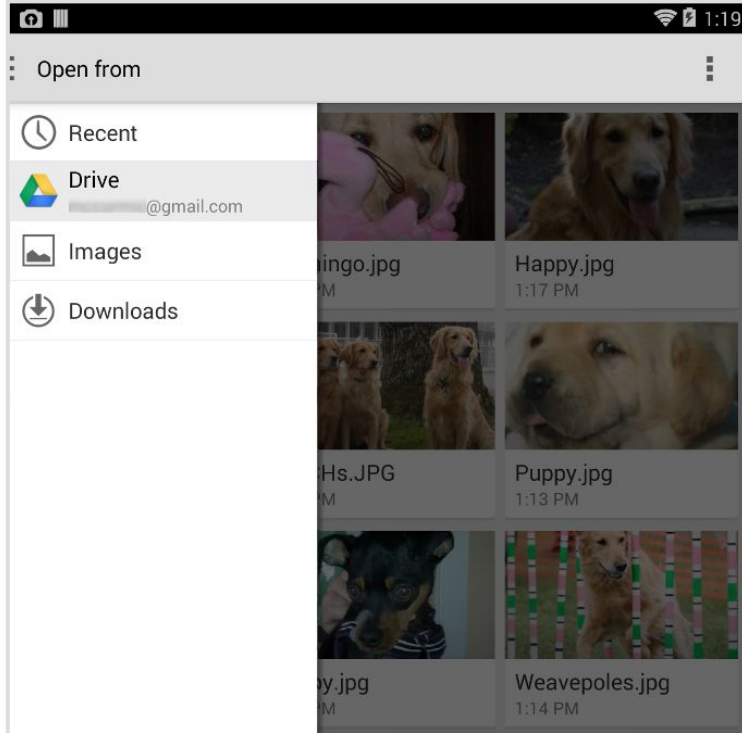
External storage

- `Environment.getExternalStoragePublicDirectory(type: String): File`
 - Type - type of files to access `Environment.DIRECTORY_*`
 - File representing top-level shared/external directory for files of particular type
 - Multi user devices - access only to current user
- `Environment.getExternalStorageFilesDir(type: String): File`
 - Type - type of files to access `Environment.DIRECTORY_*`
 - File representing where app place internal files
 - Files are deleted after app uninstall
- `Environment.isExternalStorageEmulated(): Boolean`
- `Environment.isExternalStorageRemovable(): Boolean`
- `Environment.getExternalStorageState(): String`

External storage - SD card

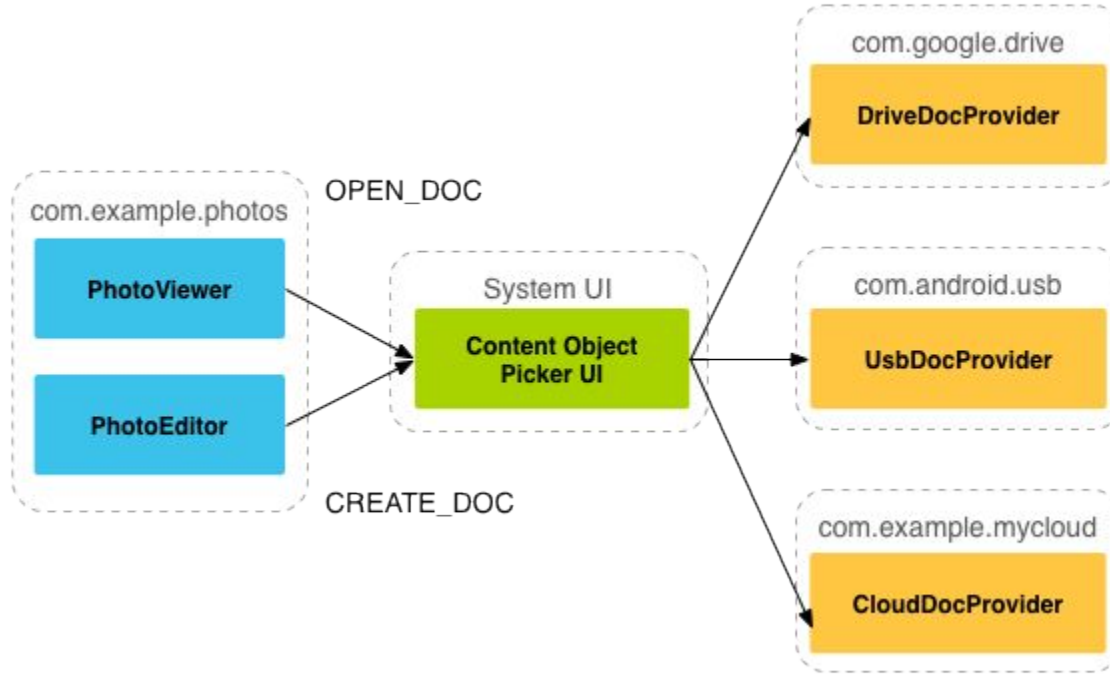
- < API-19 guess where the sdcard is mounted
- = API-19 not possible write shared data on sd card, when primary external storage is available
 - Or using storage access framework, but access is granted per file
- >=API-21 Storage access framework allows to grant access for directories
 - New APIs for accessing media folders on SD card
 - Context.getExternalMediaDirs(): Array<File>

Storage access framework



- Let user pick a file
- Allows to plug-in custom service (cloud services like Dropbox, Google drive, ...)
- Since API 19

Storage access framework



SharedPreferences

- Key value storage
- Backed by XML
- `Context.getSharedPreferences(name: String, mode: Int)`
 - Name - name of file with preferences
 - Mode - operating mode
 - `MODE_PRIVATE` - only apps with same UID have access
 - `MODE_WORLD_READABLE` - API 17 Deprecated, API 24 `SecurityException`
 - `MODE_WORLD_WRITEABLE` - API 17 Deprecated, API 24 `SecurityException`
- `Activity.getPreferences(int mode)`
 - Preferences associated with activity
- `PreferenceManager.getDefaultSharedPreferences (Context)`
 - Default preferences used by Preference framework

SharedPreferences

```
val sharedPrefs = getSharedPreferences("preferences",  
Context.MODE_PRIVATE)  
val intVal = sharedPrefs.getInt("int_key", 42)  
val stringVal = sharedPrefs.getString("string_key", "Default")
```

```
val editor = sharedPrefs.edit()  
editor.putString("string_key", "new value")  
editor.commit() //Synchronous  
editor.apply()   // Async
```

SharedPreferences

- `Editor.commit()`
 - Notifies about result
 - Synchronous operation, waits until changes are written to disk
- `Editor.apply()`
 - Async variant
 - Atomically stores values
 - ANRs bugs (`fsync()` on main thread)
- If multiple editors modifying preferences at the same time, last calling `apply()` wins
- Debugging - rooted device or flipper/stetho

<http://facebook.github.io/stetho/> - old, not maintained

<https://fbflipper.com/> - new, multiplatform

Data store

- Shared preferences replacement
- Currently alpha version
 - API can change
- Shared preferences async API is buggy

Data store

Feature	SharedPreferences	Preferences DataStore	Proto DataStore
Async API	✓ (only for reading changed values, via listener)	✓ (via <code>Flow</code>)	✓ (via <code>Flow</code>)
Synchronous API	✓ (but not safe to call on UI thread)	✗	✗
Safe to call on UI thread	✗ *	✓ (work is moved to <code>Dispatchers.IO</code> under the hood)	✓ (work is moved to <code>Dispatchers.IO</code> under the hood)
Can signal errors	✗	✓	✓
Safe from runtime exceptions	✗ **	✓	✓
Has a transactional API with strong consistency guarantees	✗	✓	✓
Handles data migration	✗	✓ (from <code>SharedPreferences</code>)	✓ (from <code>SharedPreferences</code>)
Type safety	✗	✗	✓ with Protocol Buffers

Exercise

6. Count app launches
7. Prefill login with last used one
8. Flipper for debug shared preferences

Database -SQLite

- Full-featured SQL
- Single-file database
- Source code is just 1 file
- Small footprint
- ACID transactions
- Well documented
- Supports most of the SQL92 standard

SQLite on Android

- Foreign keys disabled by default
- Internal storage
- Collation
 - BINARY - SQLite default
 - LOCALIZED - changes with system locale
 - UNICODE - Unicode collation algorithm
- Thread safe
- Create/upgrade on background thread
- Take care about opening/closing from different threads
- Use `BaseColumn._ID` for primary keys, some components rely on it
- Stetho tool for debugging

Database

- `android.database.sqlite.SQLiteOpenHelper`
 - Database creation
 - Version management
 - Sqlite configuration
 - Enable write ahead log
 - Enable support for foreign keys
- `android.database.sqlite.SQLiteDatabase`
 - Exposes methods to manage a SQLite databases
 - CRUD methods
 - Manage transactions

SQLiteOpenHelper

- `onCreate(db: SQLiteDatabase)`
 - Called when the database is created for the first time
- `onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int)`
 - Upgrade logic
- `getReadableDatabase/getWritableDatabase`
 - creates/open database
- `close()`
 - Close open database object

SQLiteDatabase

- `insert(table: String, nullColumnHack: String, values: ContentValues)`
 - Table - name of table
 - nullColumnHack - optional, allows to insert empty row
 - Values - inserted values
 - Returns ID of newly inserted row
- `long insertOrThrow`
- `long insertWithOnConflict`

SQLiteDatabase

- `query(boolean distinct,
table: String,
columns: Array<String>,
selection: String,
selectionArgs: Array<String>,
groupBy: String,
having: String,
orderBy: String,
limit: String): Cursor`
 - Selection - WHERE clause, values replaced by ?
 - selectionArgs - values to replace ? in selection
- Multiple variants of query, with different possibilities
- `rawQuery(sql: String, selectionArgs: Array<String>): Cursor`
- Close returned cursors

SQLiteDatabase

- `update(table: String,
values: ContentValues,
whereClause: String,
whereArgs: Array<String>): Int`

SQLiteDatabase

- `delete(table: String,
values: ContentValues,
whereClause: String,
whereArgs: Array<String>): Int`

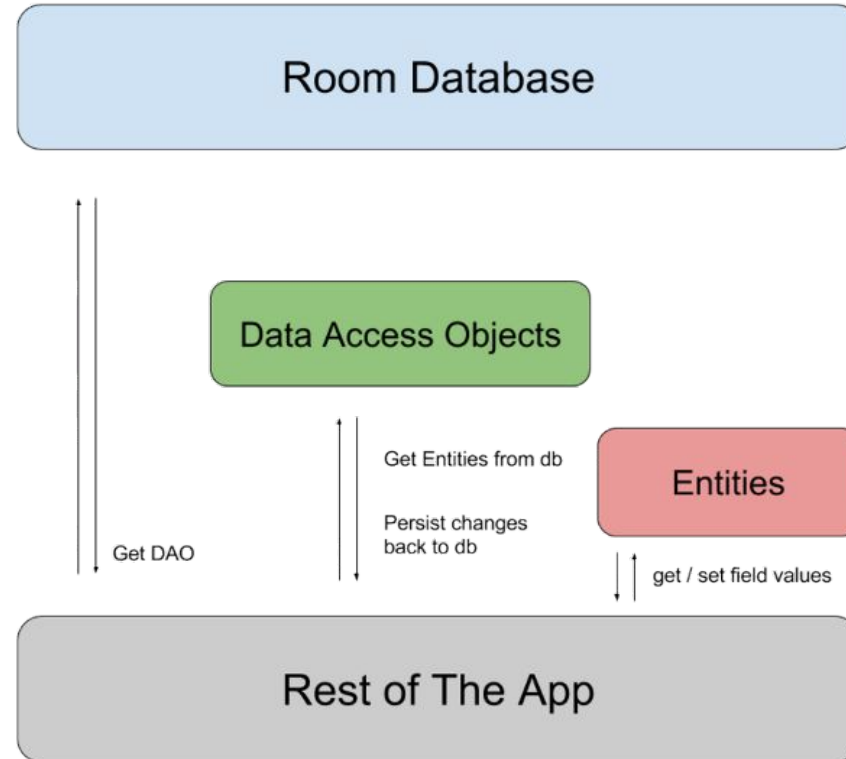
SQLiteDatabase

- Every CRUD operation is a transaction
- For inserting more rows in one time use transactions
- `beginTransaction()`
- `endTransaction()`
- `setTransactionSuccessful()`

Room

- Part of the [Android Jetpack](#)
- Abstraction over SQLite
- Compile time validation of SQL queries
- Full integration with other Architecture components (LiveData, LifecycleObserver)
- RxJava bindings

Room



Room - entities

- Represents a table

```
@Entity
data class Car(
    @PrimaryKey val id: Int,
    @ColumnInfo(name = "manufacturer") val manufacturer: String?,
    @ColumnInfo(name = "model") val model: String?,
    @ColumnInfo(name = "nubmer_of_wheels") val numberOfWheels: String?
)
```

Room - DAO

- Defines operations on top of entities

```
@Dao
interface CarDao {
    @Query("SELECT * FROM car")
    fun getAll(): List<Car>

    @Query("SELECT * FROM car WHERE id IN (:carIds)")
    fun loadAllByIds(carIds: IntArray): List<Car>

    @Query("SELECT * FROM car WHERE manufacturer LIKE :manufacturer AND " +
        "model LIKE :model LIMIT 1")
    fun findByModel(manufacturer: String, model: String): Car

    @Insert
    fun insertAll(vararg cars: Car)

    @Delete
    fun delete(car: Car)
}
```

Room database

- Defines database

```
@Database(entities = arrayOf(Car::class), version = 1)
abstract class CarDatabase : RoomDatabase() {
    abstract fun carDao(): CarDao
}
```

```
val db = Room.databaseBuilder(
    applicationContext,
    CarDatabase::class.java, "car-database"
)
    .addMigrations(...)
    .build()
```

ContentProvider

- Access to structured set of data
- Define data security
 - Via permissions
 - Global
 - Read/Write permissions
 - For single URI
- Connects data from one process to code running in another process
- ContentResolver for access data

ContentProvider

- Used by system aps
 - SMS
 - Contacts
 - Calendar
- Allows to share data between apps
- Data specified via Uri
- Allows to use CursorLoader

ContentProvider

- Can be backed up by different data sources
 - SQLite database
 - Network
 - Files
 - ...

ContentProvider

- Initializes early
 - In priority order
- Application component start order
 - Content resolvers
 - Application
 - Invoked component by intent
- <https://firebase.googleblog.com/2016/12/how-does-firebase-initialize-on-android.html>

ContentProvider - implementation

- Design data storage
- Design content URIs
 - `content://com.example.app.provider/table1`
 - `content://com.example.app.provider/table2/dataset1`
 - `content://com.example.app.provider/table3/#`
- Define UriMatcher
 - Translates Uris to number constant
- Extend ContentProvider class
 - `query()`, `insert()`, `update()`, `delete()`
 - `getType()`
 - `onCreate()` - fast operations, postpone db creation
- Register provider in manifest

ContentResolver

- `context.getContentResolver()`
- CRUD operations similar params as SQLiteDatabase
- Specify data by URI

Strict mode

Strict mode

- Developer tool
- Detects application bad behaviour

```
if (DEVELOPER_MODE) {  
    StrictMode.setThreadPolicy(StrictMode.ThreadPolicy.Builder\(\)  
        .detectDiskReads()  
        .detectDiskWrites()  
        .detectNetwork()    // or .detectAll() for all detectable problems  
        .penaltyLog()  
        .build())  
    StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder\(\)  
        .detectLeakedSqlLiteObjects()  
        .detectLeakedClosableObjects()  
        .penaltyLog()  
        .penaltyDeath()  
        .build())  
}
```



Thank you Q&A

Feedback is appreciated

prokop@avast.com

Please use [mff-android] in subject