CS 403X Mobile and Ubiquitous Computing

Lecture 9: Face Detection, Widget Catalog, SQLite Databases

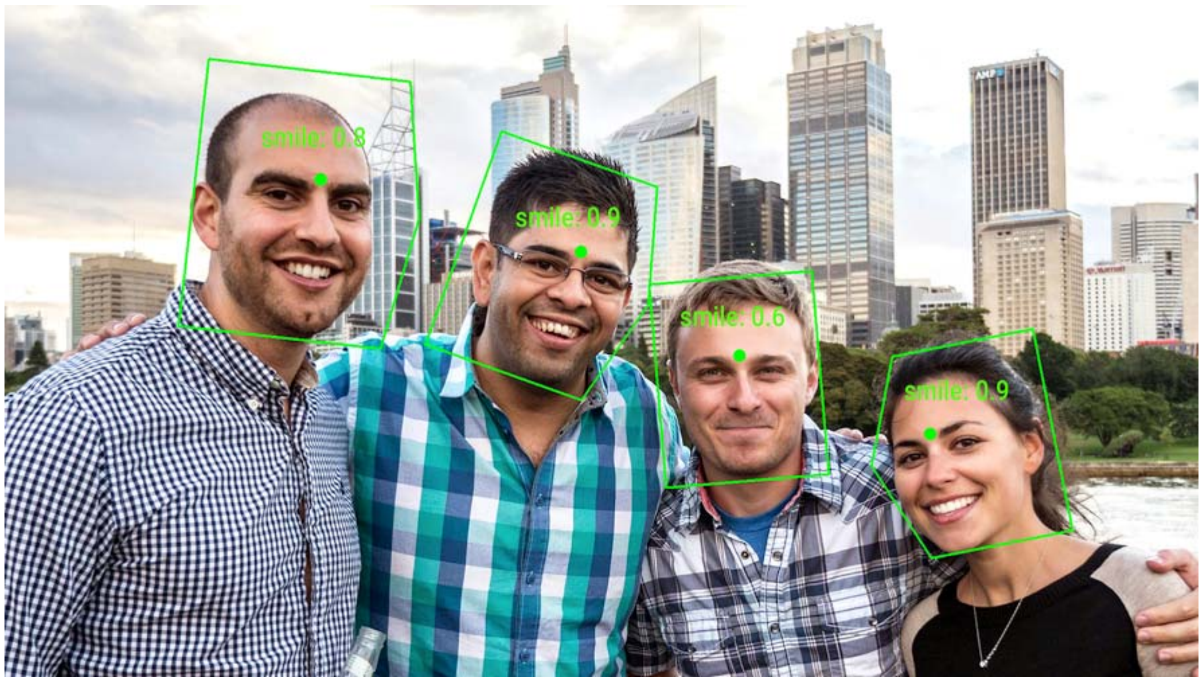
Emmanuel Agu

Face Detection

Mobile Vision API

https://developers.google.com/vision/

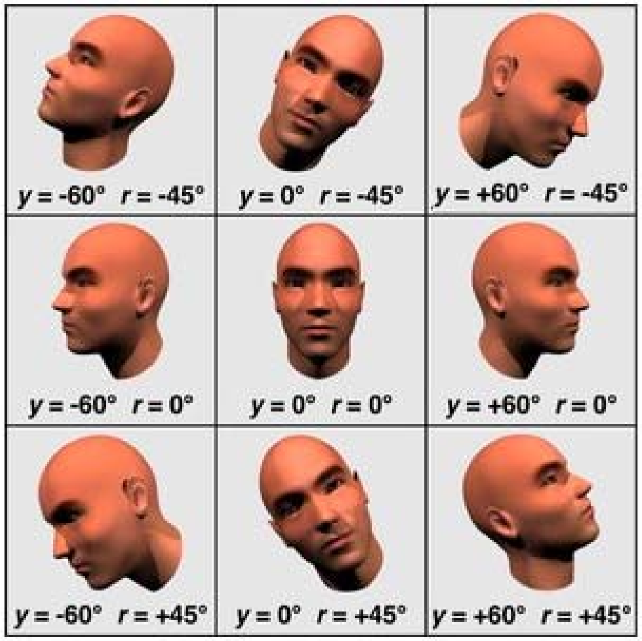
- **Face Detection:** Locate face in photos and video and
  - **Facial landmarks:** Eyes, nose and mouth
  - **State of facial features:** Eyes open? Smiling?

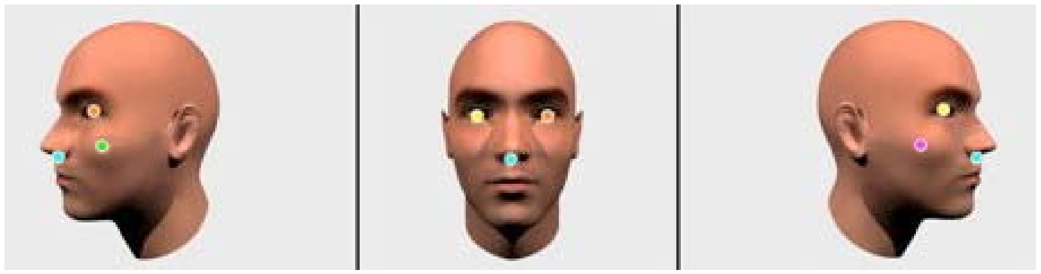


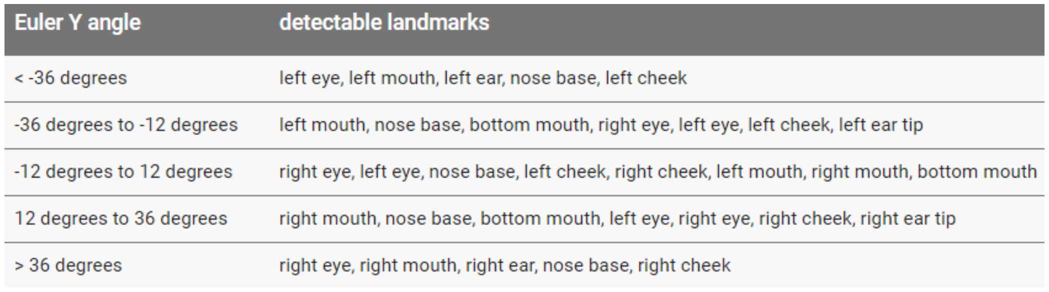
Face Detection: Google Mobile Vision API

- Detects faces that are:
  - reported at a position, with size and orientation (Euler angles)
  - Can be searched for landmarks (e.g. eyes and nose)

Landmarks







Euler Y angle	detectable landmarks
< -36 degrees	left eye, left mouth, left ear, nose base, left cheek
-36 degrees to -12 degrees	left mouth, nose base, bottom mouth, right eye, left eye, left cheek, left ear tip
-12 degrees to 12 degrees	right eye, left eye, nose base, left cheek, right cheek, left mouth, right mouth, bottom mouth
12 degrees to 36 degrees	right mouth, nose base, bottom mouth, left eye, right eye, right cheek, right ear tip
> 36 degrees	right eye, right mouth, right ear, nose base, right cheek

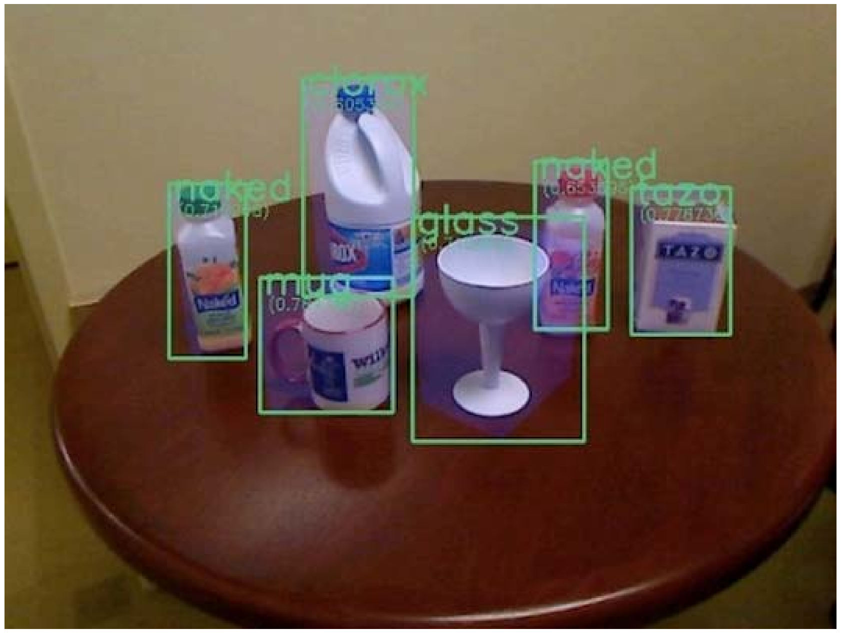
Orientation

Face Detection: Google Mobile Vision API

- Mobile Vistion API also does:
  - **Face tracking:** detects faces in consecutive video frames
  - **Classification:** Eyes open? Face smiling?
- Classification:
  - Determines whether a certain facial characteristic is present
  - API currently supports 2 classifications: eye open, smiling
  - Results expressed as a confidence that a facial characteristic is present
    - E.g. > 0.7 confidence means likely person is smiling
- Mobile vision API does detection but NOT:
  - **Face recognition:** Detects who the detected faces are (e.g. if 2 detected faces belong to the same person).

Face Detection: Google Mobile Vision API

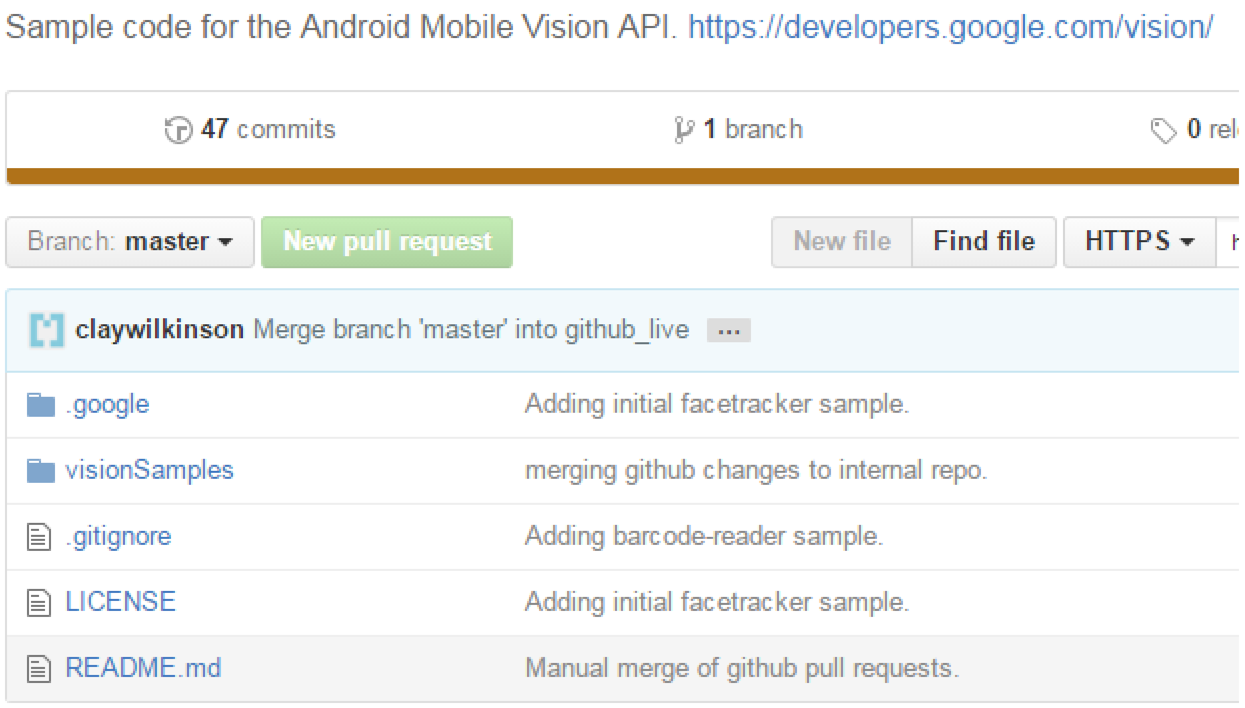
- **Face detection:** Special case of object-class detection
- **Object-class detection task:** find locations and sizes of all objects in an image that belong to a given class.
    - E.g: bottles, cups, pedestrians, and cars
- **Object matching:** Objects in picture compared to objects in database of labelled pictures



Face Detection Using Google's Mobile Vision API

Getting Started with Mobile Vision Samples

- Get **Android Play Services SDK** level 26 or greater
- Download mobile vision samples from github

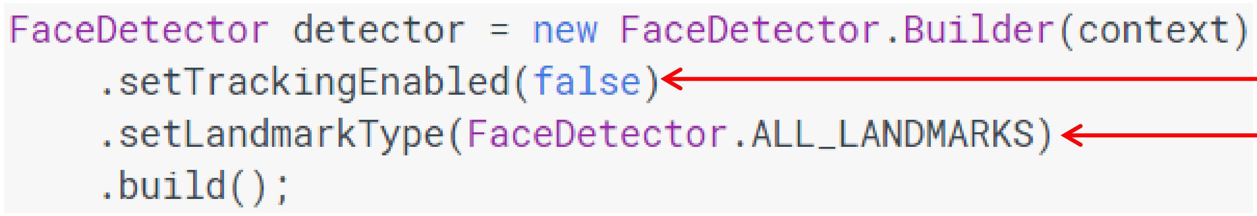

Sample code for the Android Mobile Vision API. https://developers.google.com/vision/

claywilkinson Merge branch 'master' into github_live

.google	Adding initial facetracker sample.
visionSamples	merging github changes to internal repo.
.gitignore	Adding barcode-reader sample.
LICENSE	Adding initial facetracker sample.
README.md	Manual merge of github pull requests.

Creating the Face Detector

- In app’s **onCreate** method, create face detector



```
FaceDetector detector = new FaceDetector.Builder(context)
    .setTrackingEnabled(false)
    .setLandmarkType(FaceDetector.ALL_LANDMARKS)
    .build();
```

Don’t track points

Detect all landmarks

- **detector** is base class for implementing specific detectors. E.g. face detector, bar code detector
- Tracking finds same points in multiple frames
- Detection works best in single images when **trackingEnabled** is false

Detecting Faces and Facial Landmarks

- Create Frame (image data, dimensions) instance from bitmap supplied

```
Frame frame = new Frame.Builder().setBitmap(bitmap).build();
```

- Call detector synchronously with frame to detect faces

```
SparseArray<Face> faces = detector.detect(frame);
```

- **Face** is a single detected human face in image or video
- Detector takes **Frame** as input, outputs array of **Faces**
- Iterate over array of faces, the landmarks for each face, and draw the result based on each landmark position

```
for (int i = 0; i < faces.size(); ++i) {
    Face face = faces.valueAt(i);
    for (Landmark landmark : face.getLandmarks()) {
        int cx = (int) (landmark.getPosition().x * scale);
        int cy = (int) (landmark.getPosition().y * scale);
        canvas.drawCircle(cx, cy, 10, paint);
    }
}
```

Iterate through face array

Get face at position i in Face array

Return list of face landmarks (e.g. eyes, nose)

Returns landmark's (x, y) position where (0, 0) is image's upper-left corner

Other Stuff

- To count faces, call **faces.size( )**

```
TextView faceCountView = (TextView) findViewById(R.id.face_count);
faceCountView.setText(faces.size() + " faces detected");
```

- Querying Face detector’s status

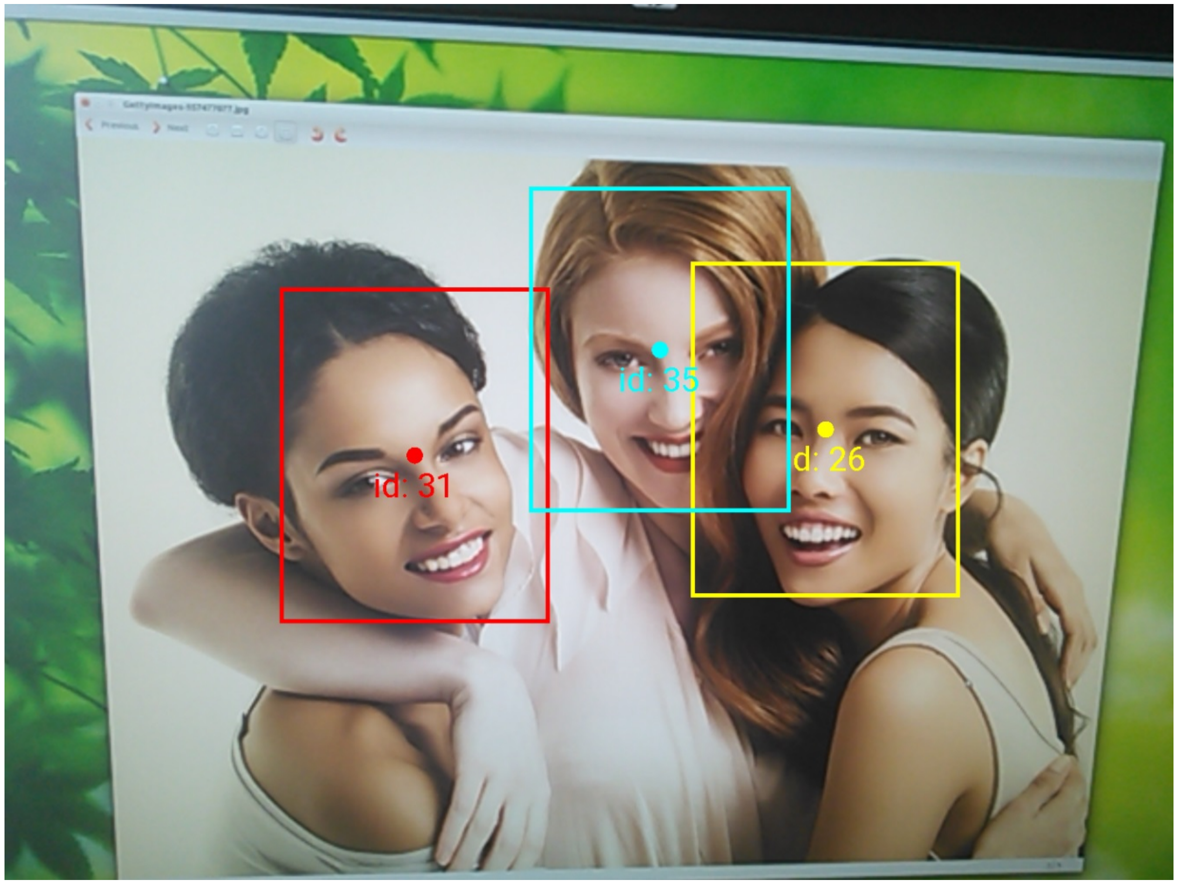
```
if (!detector.isOperational()) {
    // ...
}
```

- Releasing Face detector (frees up resources)

```
detector.release();
```

Detect & Track Multiple Faces in Video

- Can also track multiple faces in image sequences/video, draw rectangle round each one

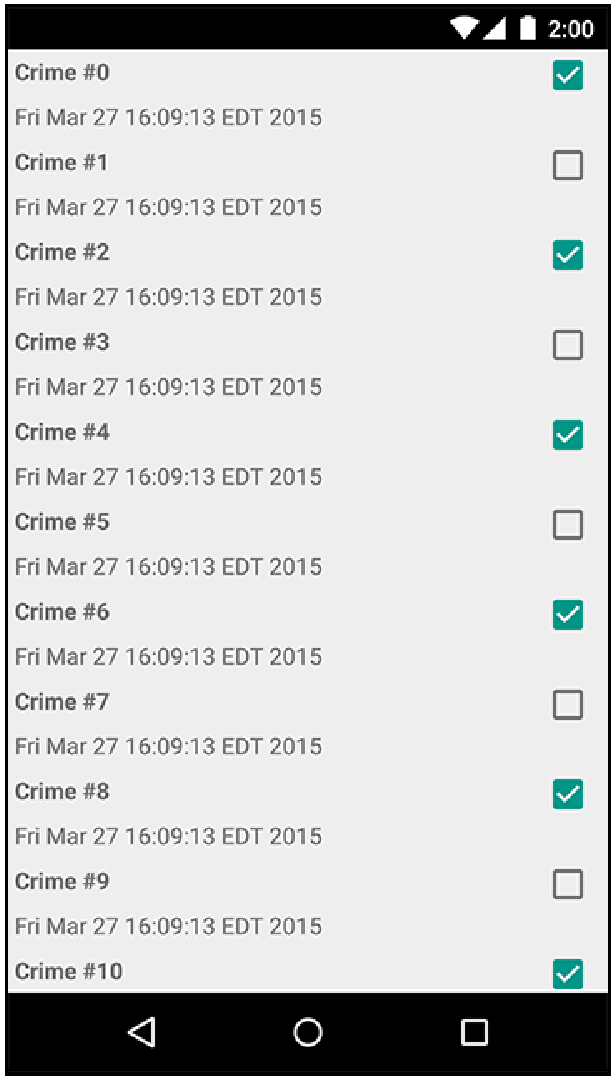


Skipped Android Nerd Ranch CriminalIntent Chapters

Chapter 9: Displaying Lists with RecyclerView

- RecyclerView facilitates view of large dataset

- E.g Allows crimes in **CriminalIntent** to be listed

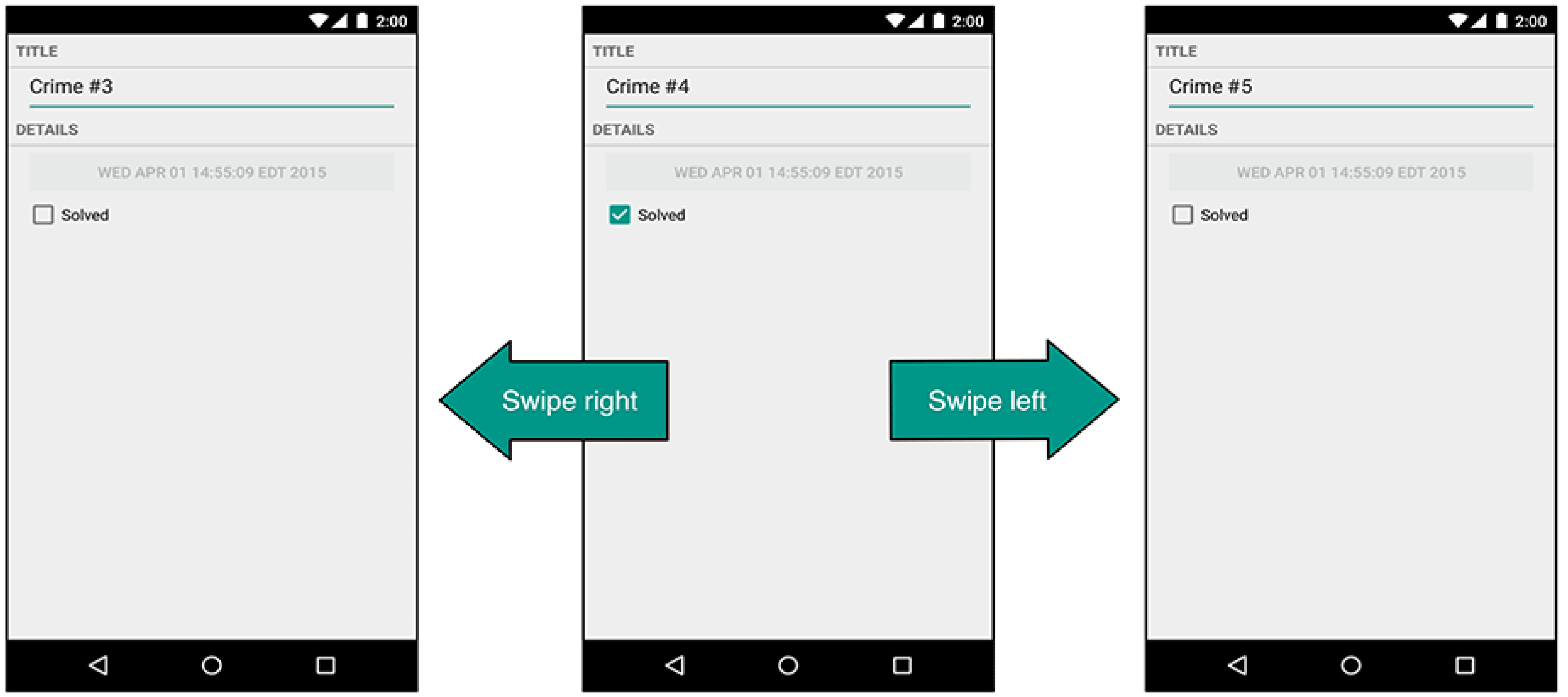


2:00

Crime #0 Fri Mar 27 16:09:13 EDT 2015	✓
Crime #1 Fri Mar 27 16:09:13 EDT 2015	☐
Crime #2 Fri Mar 27 16:09:13 EDT 2015	✓
Crime #3 Fri Mar 27 16:09:13 EDT 2015	☐
Crime #4 Fri Mar 27 16:09:13 EDT 2015	✓
Crime #5 Fri Mar 27 16:09:13 EDT 2015	☐
Crime #6 Fri Mar 27 16:09:13 EDT 2015	✓
Crime #7 Fri Mar 27 16:09:13 EDT 2015	☐
Crime #8 Fri Mar 27 16:09:13 EDT 2015	✓
Crime #9 Fri Mar 27 16:09:13 EDT 2015	☐
Crime #10	✓

Chapter 11: Using ViewPager

- ViewPager allows users swipe between screens (e.g. Tinder?)
- E.g. Users swipe between Crimes in CriminalIntent



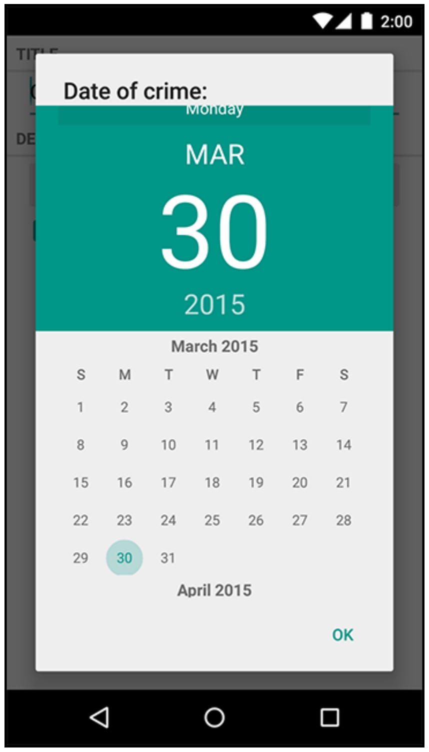
TITLE
Crime #3
DETAILS
WED APR 01 14:55:09 EDT 2015
☐ Solved

TITLE
Crime #4
DETAILS
WED APR 01 14:55:09 EDT 2015
☑ Solved

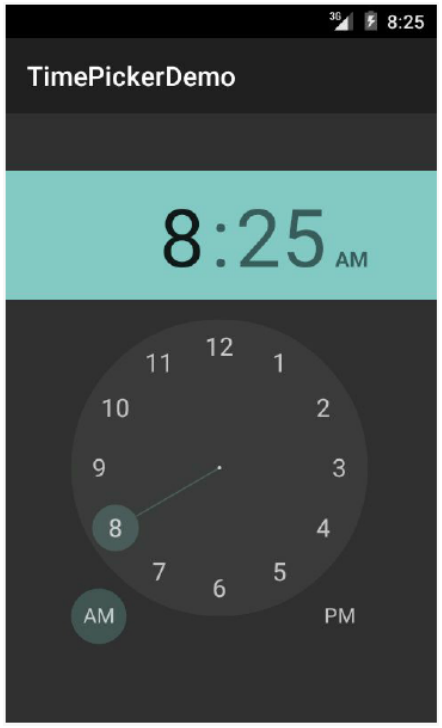
TITLE
Crime #5
DETAILS
WED APR 01 14:55:09 EDT 2015
☐ Solved

Chapter 12: Dialogs

- Dialogs present users with a choice or important information
- E.g. DatePicker allows users pick date
- Allows users to pick a date on which a crime occurred in **CriminalIntent**



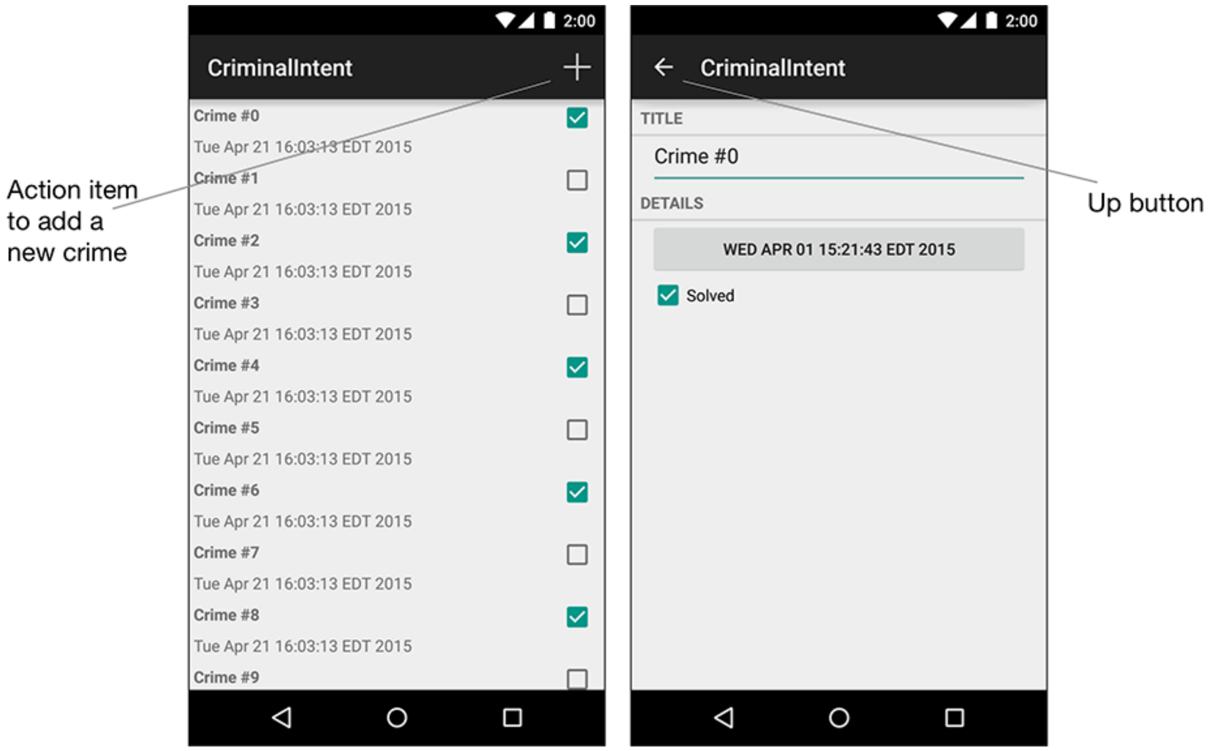
DatePicker



TimePicker

Chapter 13: The Toolbar

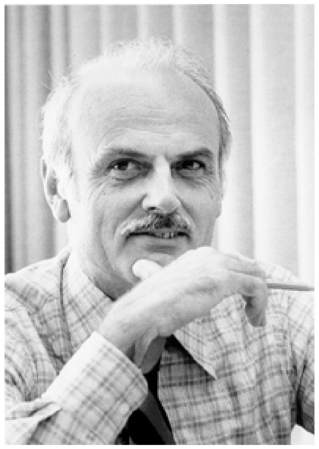
- Toolbar includes actions user can take
- In CriminalIntent, menu items for adding crime, navigate up the screen hierarchy



Android Nerd Ranch Ch 14
SQLite Databases

Background on Databases

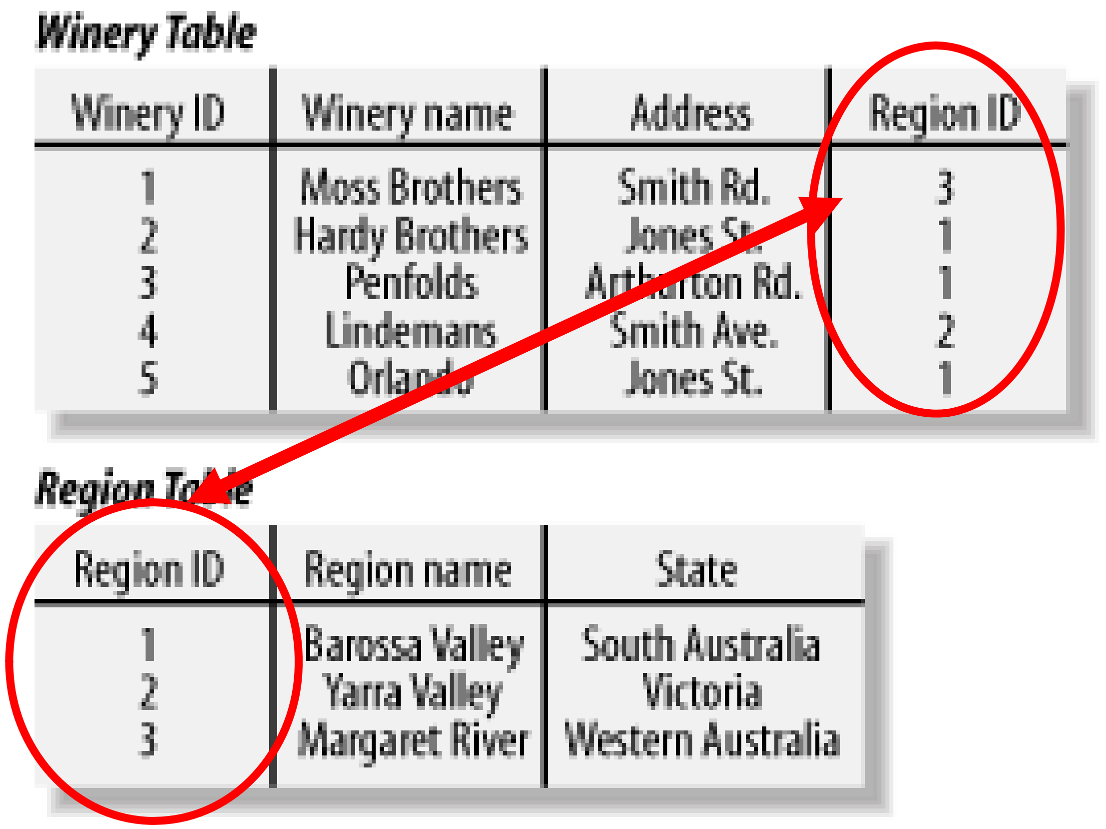
- Relational DataBase Management System (RDBMS)
  - Introduced by E. F. Codd (Turing Award Winner)



- Relational Database
  - data stored in tables
  - relationships among data stored in tables
  - data can be accessed and viewed in different ways

Example Wines Database

- **Relational Data:** Data in different tables can be related



Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

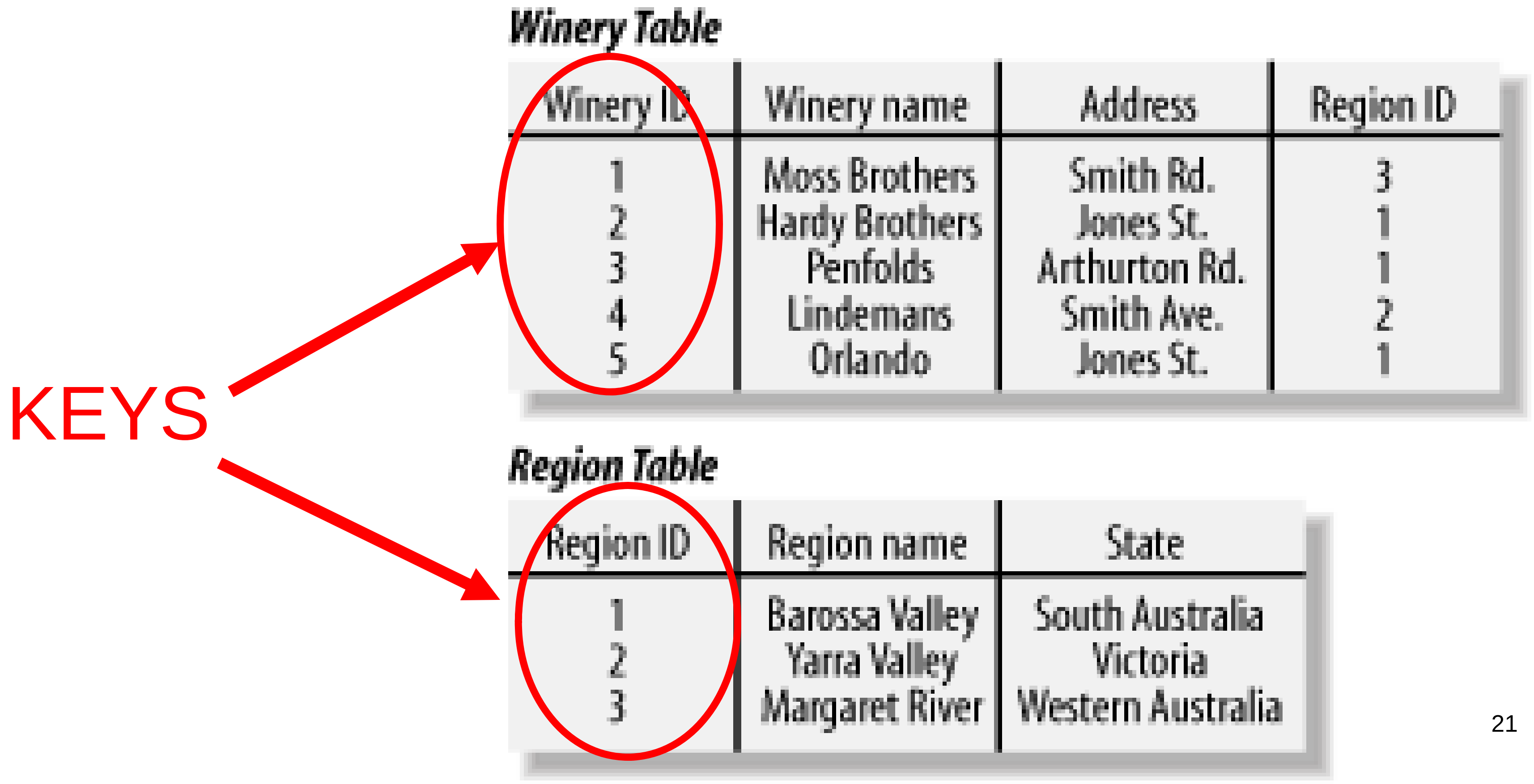
Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Ref: Web Database Applications with PHP and MySQL, 2nd Edition , by Hugh E. Williams, David Lane

Keys

- Each table has a key
- **Key:** column used to uniquely identify each row



Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

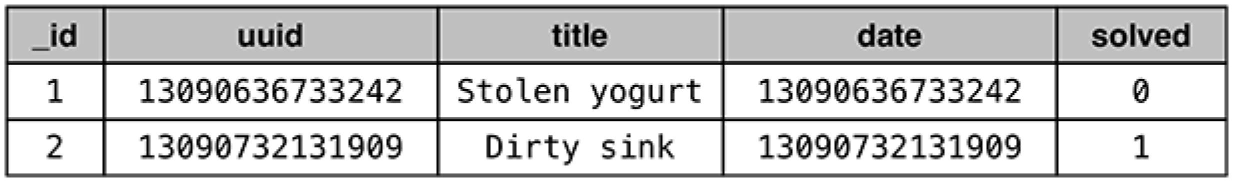
SQL and Databases

- **SQL:** language used to manipulate information in a Relational Database Management System (RDBMS)
- SQL Commands:
  - **CREATE TABLE** - creates new database table
  - **ALTER TABLE** - alters a database table
  - **DROP TABLE** - deletes a database table
  - **SELECT** - get data from a database table
  - **UPDATE** - change data in a database table
  - **DELETE** - remove data from a database table
  - **INSERT INTO** - insert new data in a database table

- SQLite implements most, but not all of SQL
  - http://www.sqlite.org/

CriminalIntent Database

- **SQLite:** open source relational database
- Android includes SQLite database
- **Goal:** Store crimes in CriminalIntent in SQLite database
- First step, define database table of **crimes**



_id	uuid	title	date	solved
1	13090636733242	Stolen yogurt	13090636733242	0
2	13090732131909	Dirty sink	13090732131909	1

CriminalIntent Database

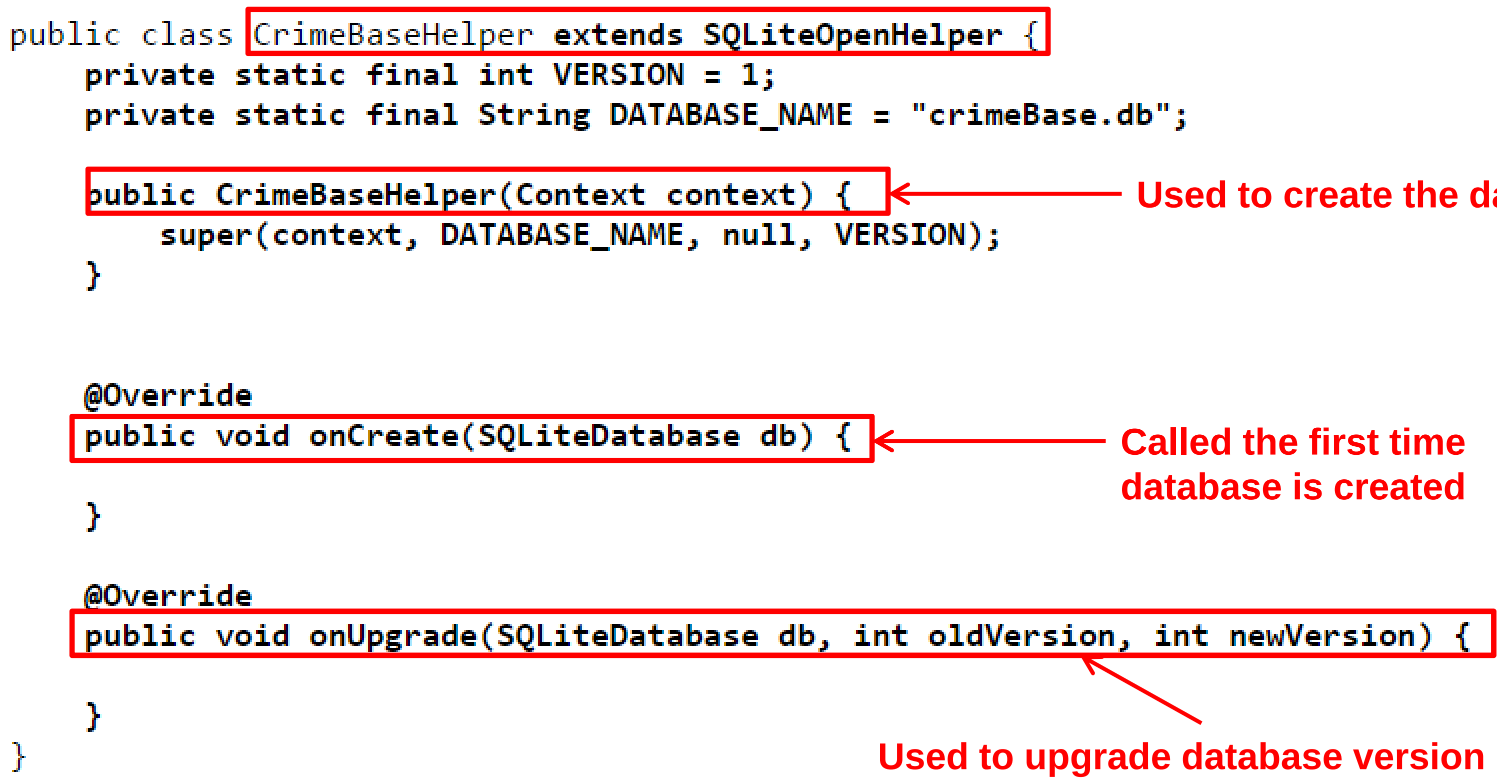
- Create **CrimeDbSchema** class to store **crime** database
- Define columns of the Crimes database table

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

_id	uuid	title	date	solved
1	13090636733242	Stolen yogurt	13090636733242	0
2	13090732131909	Dirty sink	13090732131909	1

SQLiteOpenHelper

- **SQLiteOpenHelper** class used for database creation, opening and updating
- In **CriminalIntent**, create subclass of **SQLiteOpenHelper** called **CrimeBaseHelper**



```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";

    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }


    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

Used to create the database

Called the first time database is created

Used to upgrade database version

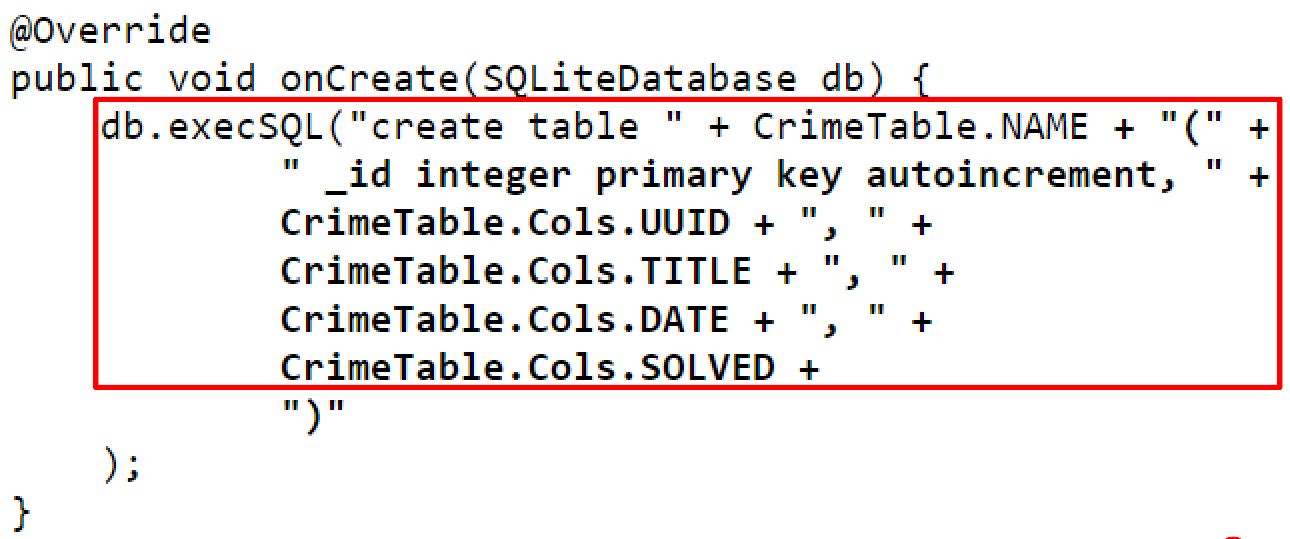
Use CrimeBaseHelper to open SQLite Database

```java
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;

    ...

    private CrimeLab(Context context) {
      mContext = context.getApplicationContext();
      mDatabase = new CrimeBaseHelper(mContext)
              .getWritableDatabase();
      mCrimes = new ArrayList<>();
    }

    ...
```

Store instance of context in variable. Will need it later

Opens new writeable Database

Create CrimeTable in onCreate( )



```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
            " _id integer primary key autoincrement, " +
            CrimeTable.Cols.UUID + ", " +
            CrimeTable.Cols.TITLE + ", " +
            CrimeTable.Cols.DATE + ", " +
            CrimeTable.Cols.SOLVED +
            ")"
    );
}
```

Create CrimeTable in our new Crimes Database

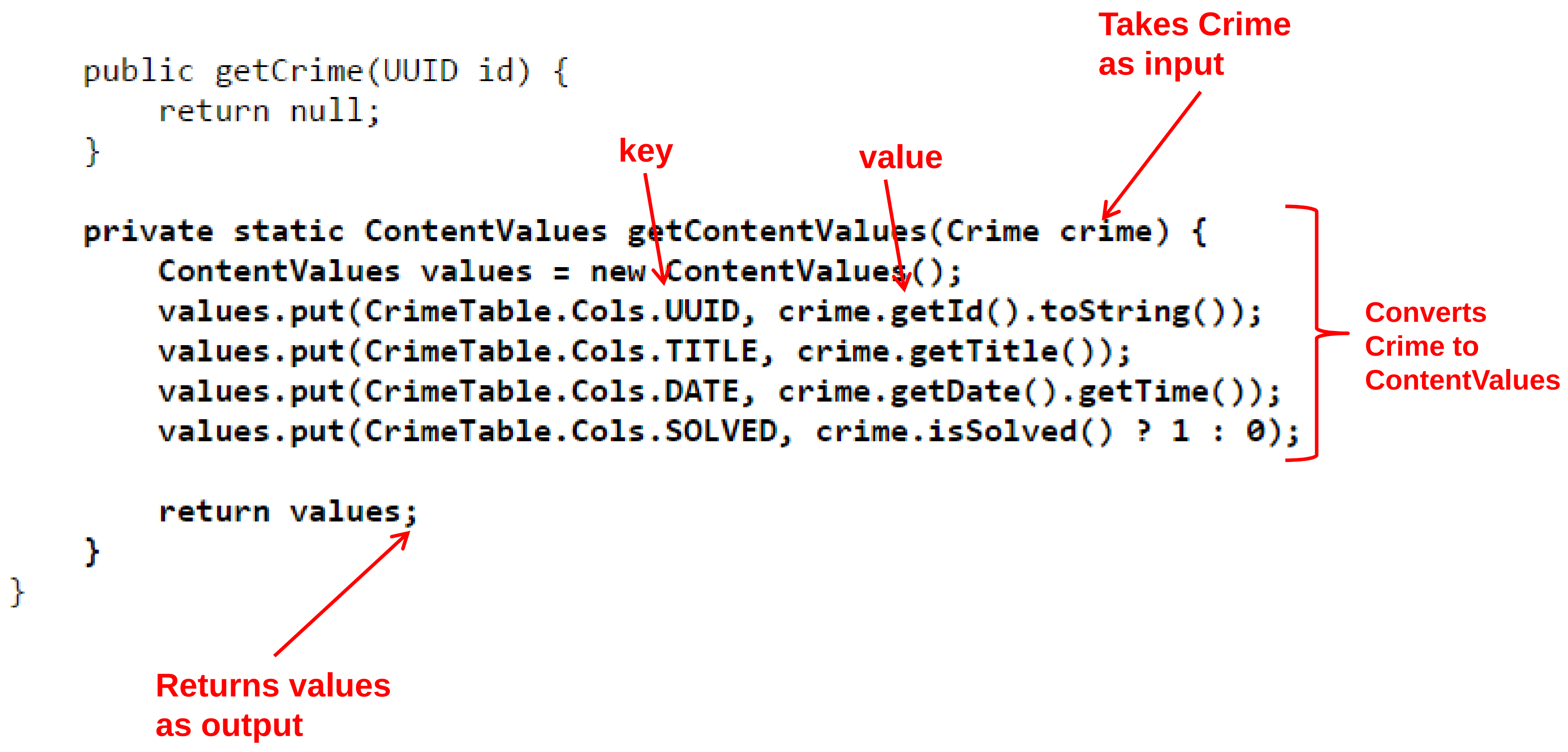
Use Database

- **CriminalIntent,** previously used arrayLists
- Modify to use SQLiteDatabase

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    ~~private List<Crime> mCrimes;~~
    private Context mContext;
    private SQLiteDatabase mDatabase;

    public static CrimeLab get(Context context) {
        ...
    }

    private CrimeLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new CrimeBaseHelper(mContext)
                .getWritableDatabase();
        ~~mCrimes = new ArrayList<>();~~
    }

    public void addCrime(Crime c) {
        ~~mCrimes.add(c);~~
    }

    public List<Crime> getCrimes() {
        ~~return mCrimes;~~
        return new ArrayList<>();
    }

    public Crime getCrime(UUID id) {
        ~~for (Crime crime : mCrimes) {~~
            ~~if (crime.getId().equals(id)) {~~
                ~~return crime;~~
            ~~}~~
        ~~}~~
        return null;
    }
}
```

Writing to the Database using ContentValues

- In Android, writing to databases is done using class **ContentValues**
- **ContentValues** is key-value pair (like Bundle)
- Create method to create **ContentValues** instance from a **Crime**



```
    public getCrime(UUID id) {
        return null;
    }

    private static ContentValues getContentValues(Crime crime) {
        ContentValues values = new ContentValues();
        values.put(CrimeTable.Cols.UUID, crime.getId().toString());
        values.put(CrimeTable.Cols.TITLE, crime.getTitle());
        values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
        values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);

        return values;
    }
}
```

Inserting Crimes in Database

- Modify **addCrime** to insert Crime into database

```
public void addCrime(Crime c) {
    ContentValues values = getContentValues(c);

    mDatabase.insert(CrimeTable.NAME, null, values);
}
```

Table you want to
Insert Crime into

ContentValue data
to insert into database

More in Text

- See Android Nerd Ranch (2nd edition), chapter 14 for the rest of the example including:
  - How to insert/update rows of the database
  - How to query the database
  - The rest of the code

Alternatives to sqlite

- SQLite is low level ("Down in the weeds“)
- Various higher level database alternatives
- E.g. Object Relational Mappers - ORM
- Higher level wrappers for dealing with sql commands and sqlite databases
- Many ORMs exist

References

- Google Mobile Vision API, https://developers.google.com/vision/
- Camera “Taking Photos Simply” Tutorials, http://developer.android.com/training/camera/photobasics.html
- Busy Coder’s guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014