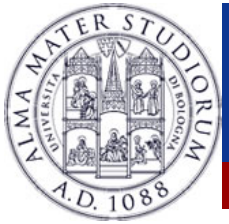
Programming with Android:
Data management

Luca Bedogni

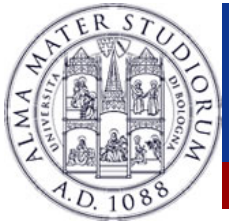
Dipartimento di Informatica: Scienza e Ingegneria

Università di Bologna



Data: outline

- Data Management in Android
  - Preferences
  - Text Files
  - XML Files
  - SQLite Database
  - Content Provider



Managing Data

Preferences: Key/Value pairs of data

Direct File I/O: Read/write files onboard or on SD cards. Remember to request permission for writing, for instance, on SD card

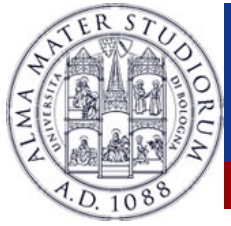
Database Tables: SQL Lite

Application Direct Access: Read only access from res assets/raw directories

Increase functionality:

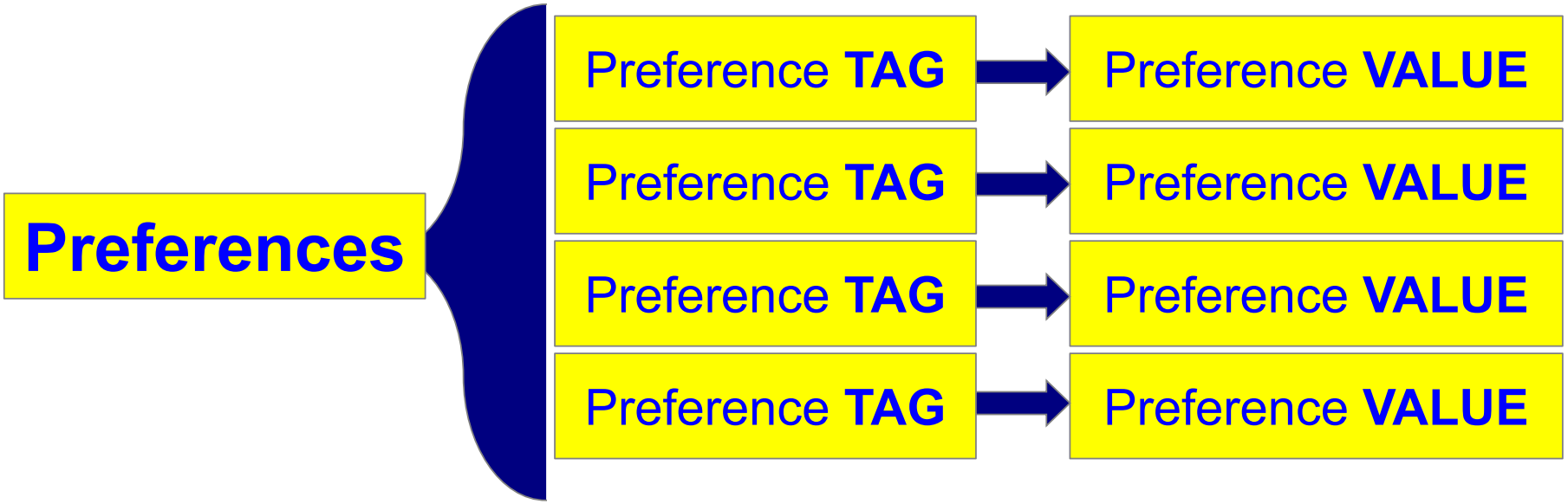
Content Providers: expose data to other applications

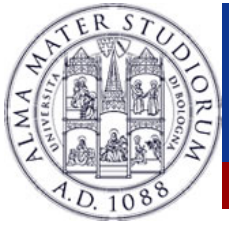
Services: background processes that run detached from any view



Preference system

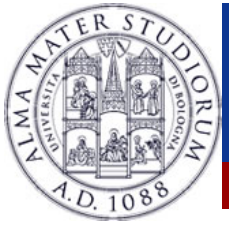
- Preferences are a convenient way to store configuration parameters
- Structured with a key-value mode





Preferences types

- Preferences could be either private or shared
  - Shared means that other applications could potentially read such preferences
  - Private means that they could be restricted at
    - Application level
    - Activity level



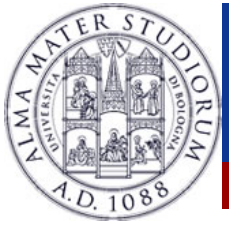
Preferences types

- Up to Android 7.0 (excluded)

```
getSharedPreferences(String name, Context.MODE_WORLD_READABLE);
getSharedPreferences(String name, Context.MODE_WORLD_WRITABLE);
```

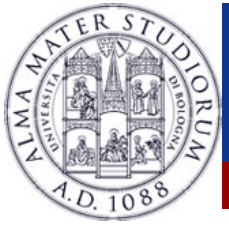
  - To share data among applications
- Starting from 7.0 it gives a Security Exception
  - To prevent, use FileProvider
- Correct method to get shared preferences

```
getSharedPreferences(String preference, Context.MODE_PRIVATE);
```



Preference example

```
public void onCreate(Bundle savedInstanceState) {
    Super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    SharedPreferences pref = getSharedPreferences(MY_TAG,
       Context.MODE_PRIVATE);
    String myData = pref.getString(MY_KEY, “No pref”);
    TextView myView = (TextView)findViewById(R.id.myTextView);
    myView.setText(myData);
}
```

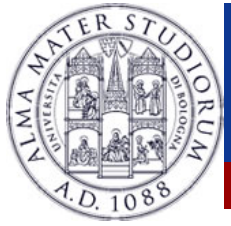


Preferences editor

- How to edit preferences?
- You need to get a SharedPreferences.Editor
- Be sure to commit operations at the end

```
SharedPreferences.Editor editor = pref.edit();
editor.putString("mydata", et.getText().toString());
editor.commit();
```

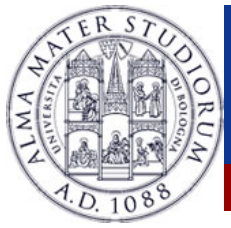
Alternatively you could also call apply() instead of commit, which writes the data to the disk asynchronously. Calling commit() stops your main thread.



Preferences screens

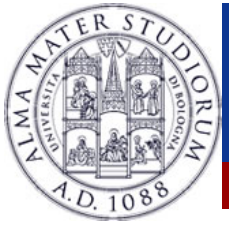
- Could be defined via XML
- Some specializations to ease the process
  - CheckBoxPreference
  - EditTextPreference
  - ListPreference
  - RingtonePreference
- Create a class that extends PreferenceActivity or PreferenceFragment and call

```
addPreferencesFromResource(R.xml.mypreferences)
```



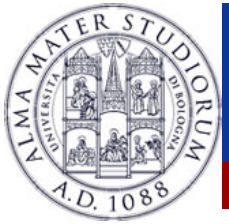
The Android FileSystem

- Linux architecture
- User privileges
  - Quite limited
- Onboard data
  - Application's reserved data
- External data
  - SD card



File I/O

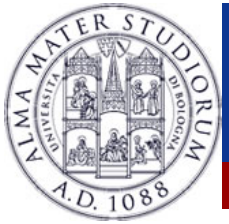
- Onboard
  - Write to a designated place for each application
  - **Where?** /data/data/<package>/files
  - **How?** Use standard java I/O classes
- SD card
  - **Where?** Environment.getExternalStorageDirectory()
  - **How?** Use standard java I/O classes
  - **Permissions?** android.permission.WRITE_EXTERNAL_STORAGE



Raw Text Files: how?

- Raw Text File
  - Place it under res/raw/ directory
  - Fill it with the text you like
  - Cannot edit it
  - Get its content

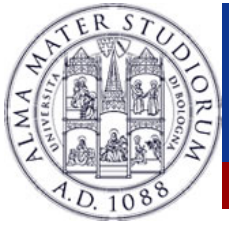
```
InputStream file = getResources().openRawResource(R.raw.myfile);
```



XML Files: how?

- XML File
  - Place it under res/xml/ directory
  - Start the file with

    <?xml version="1.0" encoding="utf-8"?>
  - Add whatever you want with <mytag>value</mytag>



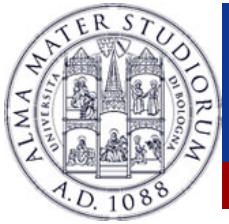
XML Files: example

- We want to visualize all the grades of this class
- Our XML file is like this:

```
<student
    name="Student's name"
    class="Laboratorio di Applicazioni Mobili"
    year="2014"
    grade="30L" />
```

XML Files: code example

```
XmlResourceParser grades = getResources().getXml(R.xml.myxmlfile);
LinearLayout ll = (LinearLayout)findViewById(R.id.myLL);int tag = -1;
while (tag != XmlResourceParser.END_DOCUMENT) {
if (tag == XmlResourceParser.START_TAG) {
      String name = grades.getName();
            if (name.equals("student")) {
                  TextView tv = new TextView(this);
                  LayoutParams lp = new LayoutParams(LayoutParams.FILL_PARENT,
               LayoutParams.WRAP_CONTENT);
               tv.setLayoutParams(lp);
                  String toWrite = grades.getAttributeValue(null, "name") + …;
                  tv.setText(toWrite); ll.addView(tv);
            }     }
            try {       tag = grades.next();      } catch (Exception e) { }
      }
```

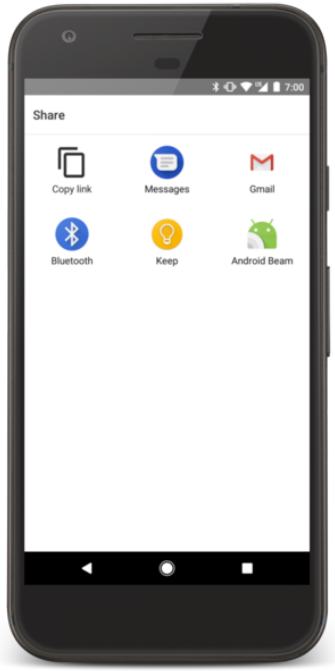


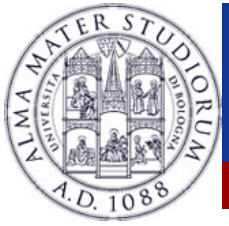
Sharing Data

If your APP wants to share data, it can do so with Intents

Specifically ACTION_SEND advertises that an app is sending data to something else

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(Intent.createChooser(sendIntent,
        getResources().getText(R.string.send_to)));
```

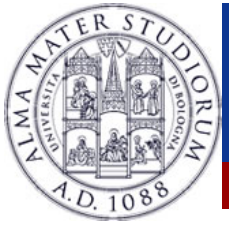




Sharing data (even easier!)

- Starting from Android 4.0 (API 14), use an ActionProvider
  - Once attached to a menu item, hanndles both appearance and behavior

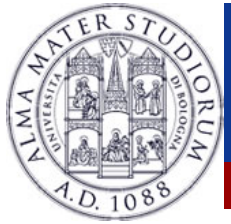
```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item
         android:id="@+id/menu_item_share"
         android:showAsAction="ifRoom"
         android:title="Share"
         android:actionProviderClass=
            "android.widget.ShareActionProvider" />
   ...
</menu>
```



Sharing data (even easier!)

- You also need the appropriate ShareIntent
  - Once attached to a menu item, hanndles both appearance and behavior

```
private ShareActionProvider mShareActionProvider;

public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.share_menu, menu);
    MenuItem item = menu.findItem(R.id.menu_item_share);

    mShareActionProvider = (ShareActionProvider) item.getActionProvider();
    return true;
}

private void setShareIntent(Intent shareIntent) {
        mShareActionProvider.setShareIntent(shareIntent);
}
```



SQLite

General purpose solution

- Lightweight database based on SQL

Standard SQL syntax

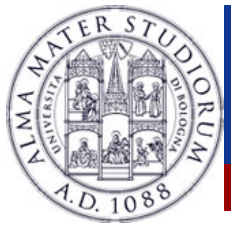
```
SELECT name FROM table WHERE name = “Luca”
```

Android gives a standard interface to SQL tables of other apps

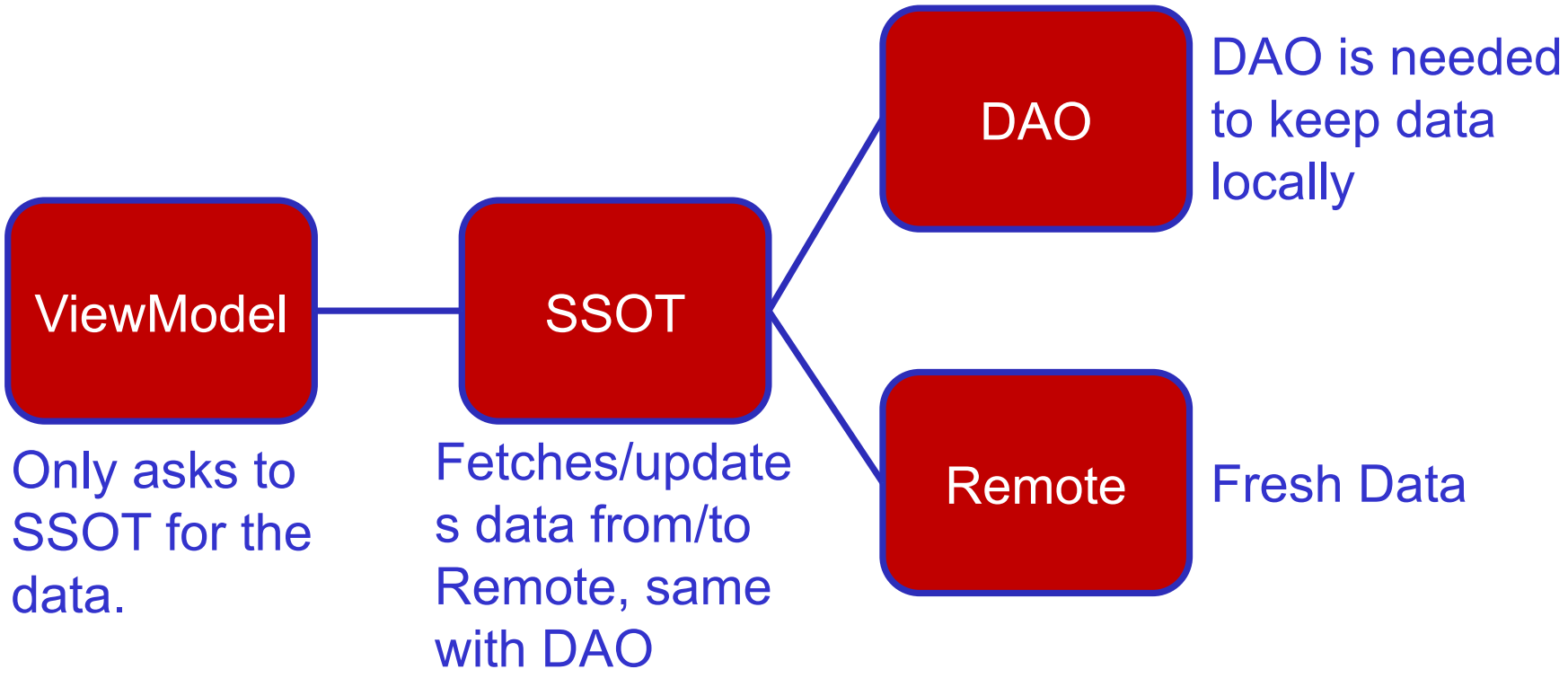
For application tables no content providers are needed

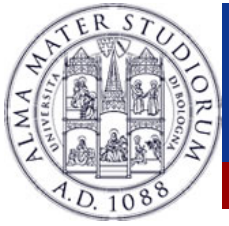
Why a local database? **Since you can’t assume connectivity**

Single Source of Truth: ***SSOT refers to the concept where certain data has only one official source to be used by data consumers (i.e. humans and software) for the true current version of that data.***



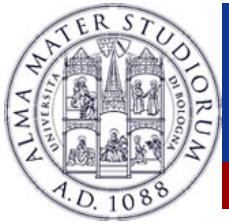
SSOT model


ViewModel
SSOT
DAO
Remote
DAO is needed to keep data locally
Fresh Data
Only asks to SSOT for the data.
Fetches/updates data from/to Remote, same with DAO



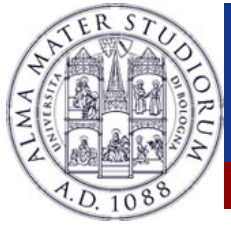
SQLite: how?

- A database to store information
- Useful for structured informations
- Create a DBHelper which extends SQLiteOpenHelper
- Fill it with methods for managing the database
  - Better to use constants like
    - TABLE_GRADES
    - COLUMN_NAME
    - ….



SQLite: example

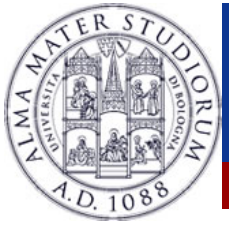
- Our database will look like this:
  - grade table:
    - id: integer, primary key, auto increment
    - firstName: text, not null
    - lastName: text, not null
    - class: text, not null
    - grade: integer, not null



SQLite: better to use constants

- Useful for query definition
- Our constants?

```
private static final String DB_NAME = “grades.db”;
private static final int DB_VERSION = 1;
public static final String TABLE_GRADES = “grades”;
public static final String COL_ID = “id”;
public static final String COL_FIRSTNAME = “firstName”;
public static final String COL_LASTNAME = “lastName”;
public static final String COL_CLASS = “class”;
public static final String COL_GRADE = “grade”;
```



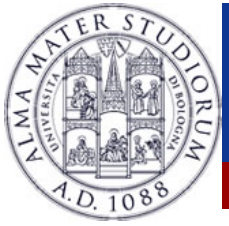
SQLite: creation code

- Constructor: call the superconstructor

```
Public mySQLiteHelper(Context context) {
    super(context, DB_NAME, null, DB_VERSION);
}
```

- onCreate(SQLiteDatabase db): create the tables

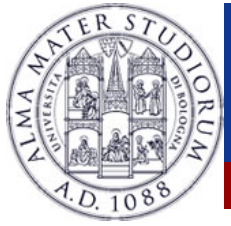
```
String sql_grade = “create table “ + TABLE_GRADES + “( “ +
    COL_ID + “ integer primary key autoincrement, “ +
    COL_FIRSTNAME + “ text not null, “ +
    COL_LASTNAME + “ text not null, “ +
    COL_CLASS + “ text not null, “ +
    COL_GRADE + “ text not null “);”;
db.execSQL(sql_grade);
```



SQLite: insert code

- Create a public method, like insertDb(…)

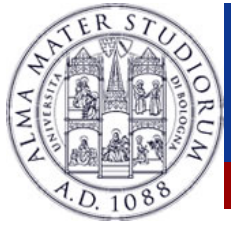
```
mySQLiteHelper sql = new mySQLiteHelper(InsertActivity.this);
SQLiteDatabase db = mySQLiteHelper.getWritableDatabase();
ContentValues cv = new ContentValues();
cv.put(mySQLiteHelper.COL_FIRSTNAME, firstName);
cv.put(mySQLiteHelper.COL_LASTNAME, lastName);
cv.put(mySQLiteHelper.COL_CLASS, className);
cv.put(mySQLiteHelper.COL_GRADE, grade);

long id = db.insert(mySQLiteHelper.TABLE_GRADES, null, cv);
```



SQLite: delete code

- Create a public method, like deleteDb(…)
- The delete method returns the number of rows affected
- Example:

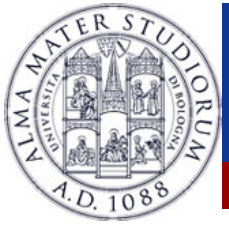
```
db.delete(mySQLiteHelper.TABLE_GRADES, “id = ?”, new   String[]
{Integer.toString(id_to_delete)});
```



SQLite: update code

- Create a public method, like updateDb(…)

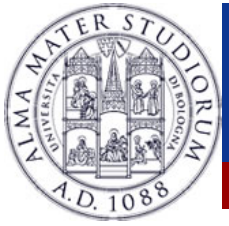
```
ContentValues cv = new ContentValues();
values.put(mySQLiteHelper.FIRSTNAME, firstName);
values.put(mySQLiteHelper.LASTNAME, lastName);

db.update(mySQLiteHelper.TABLE_GRADES, values, “id = ?”, new  String[]
{Integer.toString(id_to_update));
```



SQLite: search code

- Create a public method, like getFromDb(…)

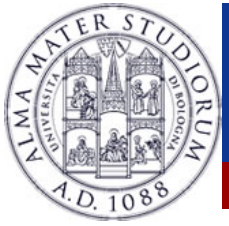
```
Cursor gradeCursor = db.query(mySQLiteHelper.TABLE_GRADES,
    new String[]{mySQLiteHelper.COL_GRADE}, mySQLiteHelper.COL_ID + " = "
    + id_to_search_for, null, null, null, null);
```



Cursors: data handlers

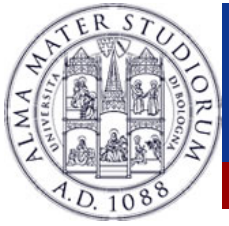
- A Cursor stores data given by a DB query
- Some methods:
  - getCount()
  - moveTo{First,Next,Last,Position,Previous}()
  - close()
- You need to look inside the Cursor to see query's results

```
while (gradeCursor.moveToNext()) {
      Log.v("GRADES",gradeCursor.getString(0));
}
```



Cursors: methods

- Manipulating the cursor
  - cursor.moveToFirst()
  - while (cursor.moveToNext())
  - for (cursor.moveToFirst(); !cursor.isAfterLast(); cursor.moveToNext())
- Get column numbers from names
  - int nameColumn = cursor.getColumnIndex(People.NAME);
  - int phoneColumn = cursor.getColumnIndex(People.NUMBER);
- Get Data from column
  - String name = cursor.getString(nameColumn);
  - String number = cursor.getString(phoneColumn);

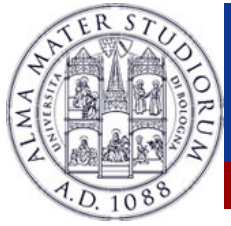


Databases with Room

- Room provides an abstraction layer over SQLite
  - You should always use Room from now on

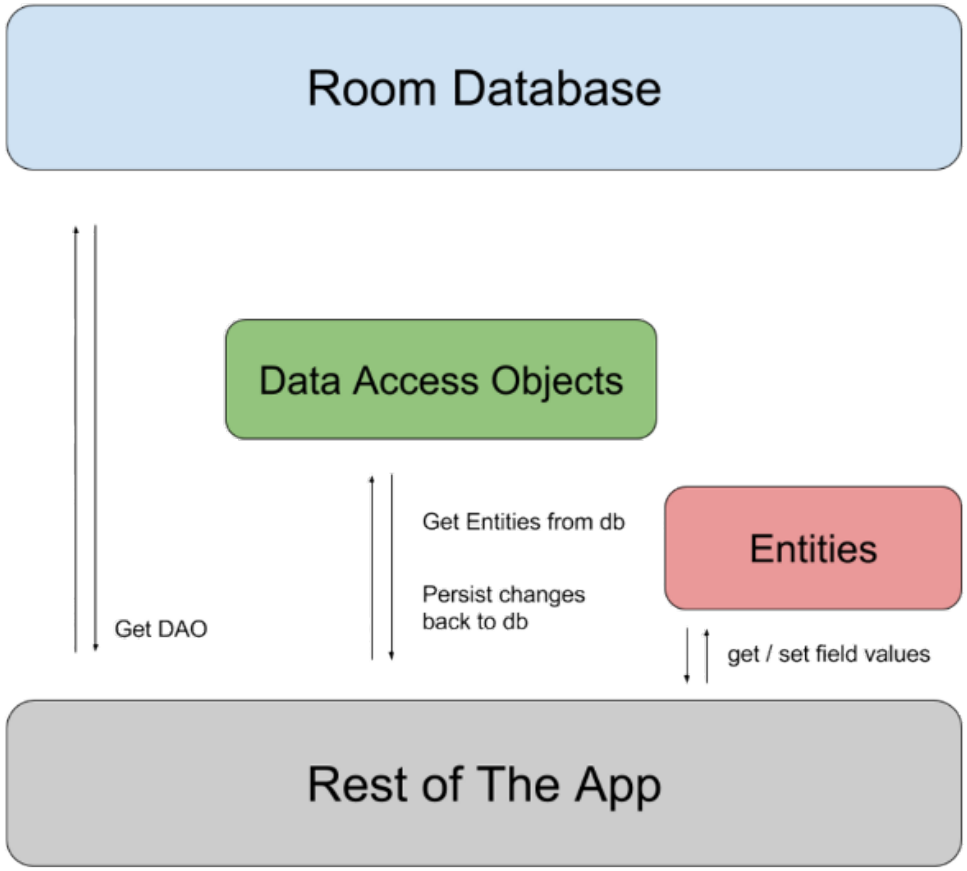
- Add it to your APP by adding in build.gradle

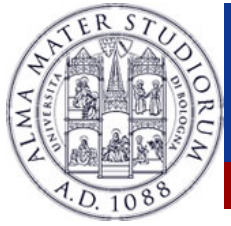
```
dependencies {
    implementation "android.arch.persistence.room:runtime:1.0.0"
    annotationProcessor "android.arch.persistence.room:compiler:1.0.0"
}
```



Room architecture

- Database
  - Contains the database holder
  - Main access point
- Data Access Objects (DAOs)
  - Interface with methods to access the database
- Entities
  - Database tables



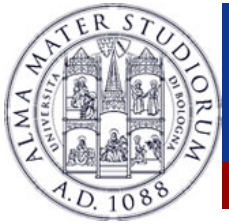


Room components: Database

It has to be an abstract class

```
@Database(version = 1, entities = {Entity1.class, Entity2.class})
abstract class myDatabase extends RoomDatabase {
abstract public Entity1Dao entity1Dao();
abstract public Entity2Dao entity2Dao();
abstract public twoEntitiesDao twoEntitiesDao();
}
```

It handles automatically the conversion from a Cursor to your APP classes

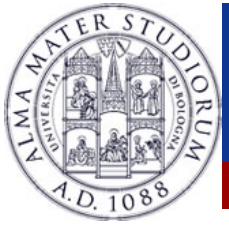


Room components: Entity

For each Entity, Room creates a database Table

Each field referencces a column, except for those marked with @Ignore

```
@Entity
class Entity1 {
    @PrimaryKey
    public int myId;
    public String firstField;
    public String secondField;

    @Ignore
    String tmp;
}
```



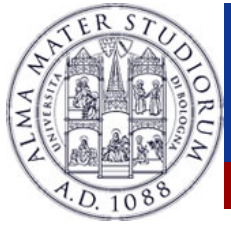
Room components: Entity

- Entities fields needs to be either public or you have to provide getters and setters
- Each entity needs at least one @PrimaryKey
  - Primary keys can be defined with more than one field

```
@Entity(primaryKeys = {"firstName", "lastName"})
```

  - The autoGenerate property automatically assigns IDs

```
@PrimaryKey(autoGenerate = true)
private int uid;
```



Room components: Entity

- Room creates a table with the Entity name
  - Change it with

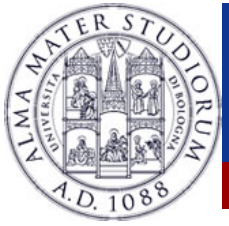
```
@Entity(tableName = "users")
```

  - Same goes for the columns

```
@ColumnInfo(name = "first_name")
    public String firstName;

@ColumnInfo(name = "last_name")
    public String lastName;
```

- Speed up queries with Indices

```
@Entity(indices = {@Index("name"), @Index(value = {”first_name", ”last_name"})})
```



Room components: Entity

- Defining uniqueness

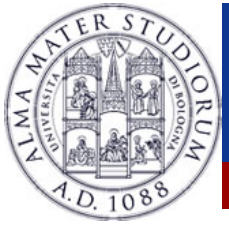
```
@Entity(indices = {@Index(value = {"first_name", "last_name"}, unique = true)})
```

- Defining relationships

```
@Entity(foreignKeys = @ForeignKey(entity = User.class,
                                  parentColumns = "id",
                                  childColumns = "user_id"))
```

- Nested objects

```
Class Material {
        public String name;
        public String weight;
}

@Entity
Class myEntity {
        …
        @Embedded
        public Material objectMaterial;
}
```

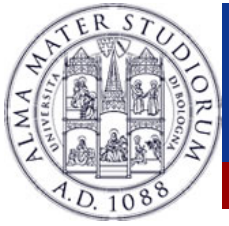


Room components: DAO

- You need DAOs to access data
- A DAO can be either an interface or an abstract class
- Room creates DAO implementations at compile time
- Syntax

```
@Dao
public interface MyDao {
    @QueryType(params..)
    public void method(method parameters);
}
```

- @QueryType can be:
  - @Insert, @Update, @Delete, @Query



DAO: Query examples

- @Insert

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
public void insertUsers(User... users);

@Insert
public void insertBothUsers(User user1, User user2);

@Insert
public void insertUsersAndFriends(User user, List<User> friends);
```

- @Update

```
@Update
public void updateUsers(User... users);
```

- @Delete

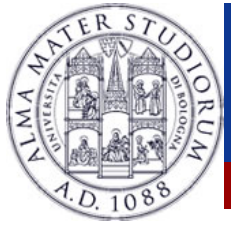
```
@Delete
public void deleteUsers(User... users);
```

- @Query

```
@Query("SELECT * FROM user")
public User[] loadAllUsers();
```

- @Query + parameters

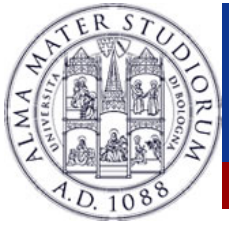
```
@Query("SELECT * FROM user WHERE age > :minAge")
public User[] loadAllUsersOlderThan(int minAge);
```



Room: migrating databases

- Updating APP's features may require updating the database
  - You add a UI field and need to add a DB field
  - You change the type of a field
  - You don't need anymore a field
- Room handles it providing the Migration environment
  - Remember:

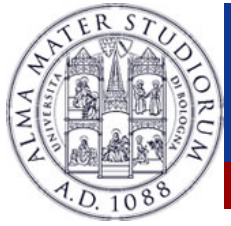
```
@Database(version = 1, entities = {Entity1.class, Entity2.class})
abstract class myDatabase extends RoomDatabase {
…
}
```



Room: migrating databases

- Each Migration class defines a startVersion and endVersion
  - At runtime, Room runs each migrate method in order

```
Room.databaseBuilder(getApplicationContext(), MyDb.class, "database-name")
        .addMigrations(MIGRATION_1_2, MIGRATION_2_3).build();

static final Migration MIGRATION_1_2 = new Migration(1, 2) {
    @Override
    public void migrate(SupportSQLiteDatabase database) {
        database.execSQL("CREATE TABLE `Fruit` (`id` INTEGER, "
                + "`name` TEXT, PRIMARY KEY(`id`))");
    }
};

static final Migration MIGRATION_2_3 = new Migration(2, 3) {
    @Override
    public void migrate(SupportSQLiteDatabase database) {
        database.execSQL("ALTER TABLE Book "
                + " ADD COLUMN pub_year INTEGER");
    }
};
```

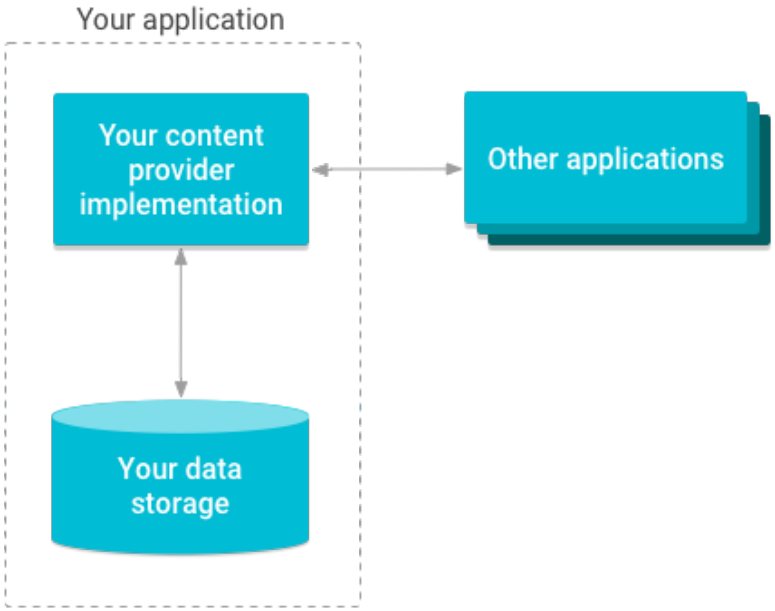


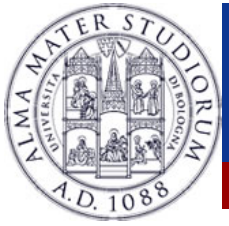
Content Providers

- A system to access shared data
- Similar to a REST web service
- For each Content Provider, one or more URIs are assigned in the form:

  content://<authority>/path

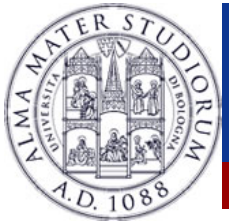
- Be aware that some ContentProviders may rquest permissions





To build a Content Provider

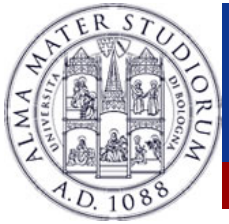
- Define the DB
- Create a class that extends android.content.ContentProvider
- Implement query(), insert(), update(), delete(), getType(), onCreate()
- Register the ContentProvider in the manifest



Sharing Files

- You can easily share files using FileProvider
- Add an entry in the manifest

```
<application
      ...>
      <provider
         android:name="android.support.v4.content.FileProvider"
         android:authorities="com.example.myapp.fileprovider"
         android:grantUriPermissions="true"
         android:exported="false">
         <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/filepaths" />
      </provider>
      ...
   </application>
```



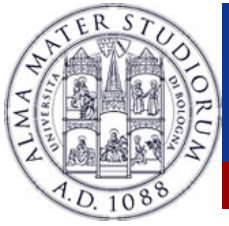
Sharing Files

- Create res/xml/filepaths.xml

```
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

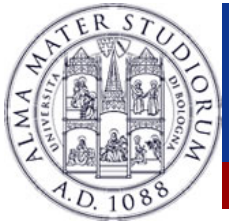
- Now other apps can access your file using URI like

```
content://com.example.myapp.fileprovider/myimages/default_image.jpg
```



How to use a Content Provider

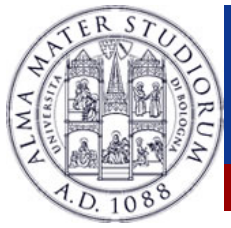
- Need to get the URI
  - Usually this is declared as public inside the content provider class
- Make a query, maybe adding some where clauses
  - You'll get a Cursor after that
- Navigate the Cursor



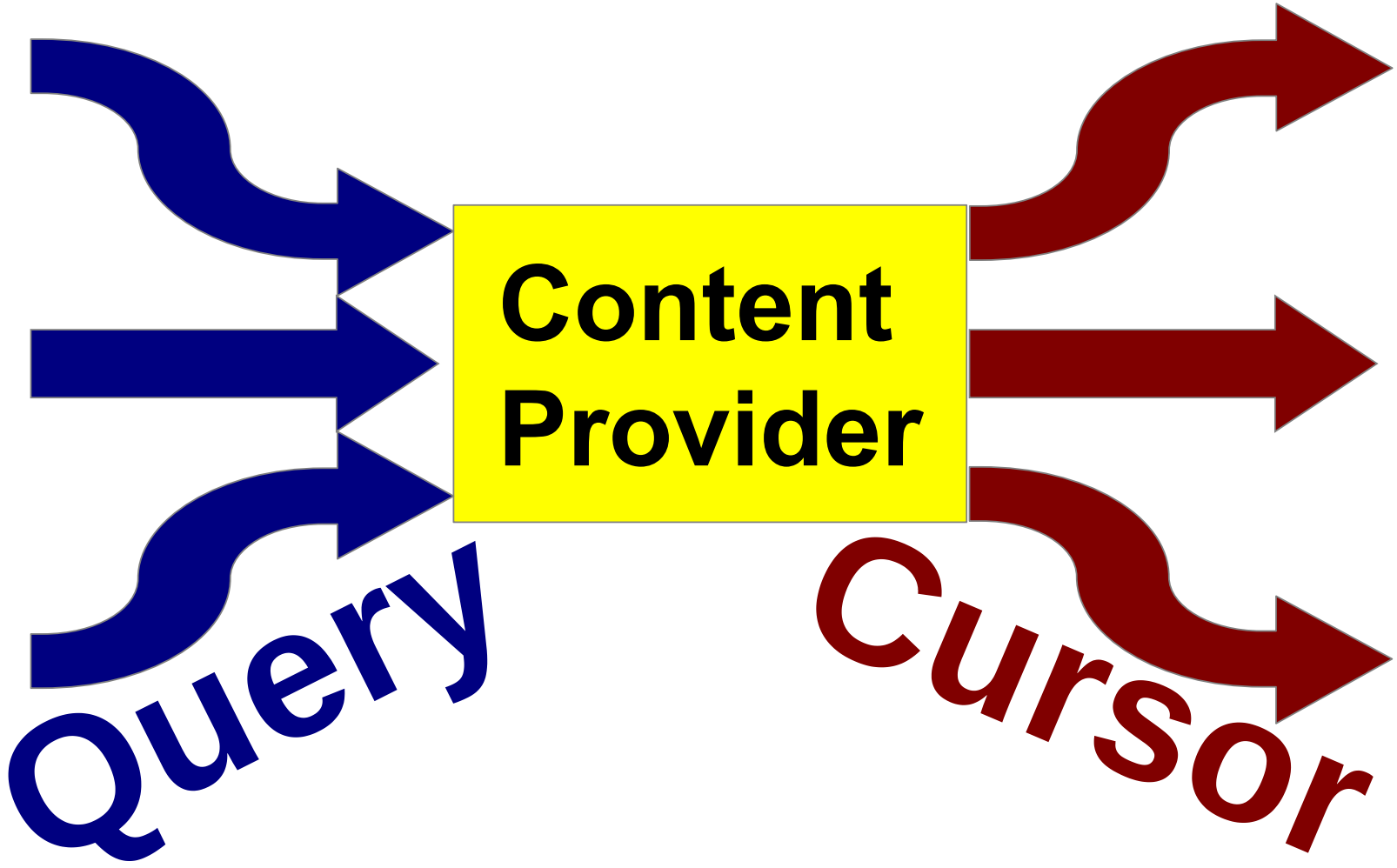
Android Content Providers

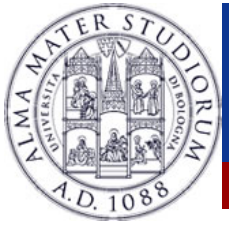
Class	Description
AlarmClock	To interact with the alarm
BlockedNumberContract	To get blocked numbers
Browser	To perform commands on the browser
CalendarContract	To handle calendar information
CallLog	Log of past calls
ContactsContract	Get and add contacts
DocumentsContract	Interact with documents
DocumentsProvider	Interact with documents
MediaStore	Access Video, Pictures, Audio and more
Settings	Inquiry system settings
…	

Find them all at https://developer.android.com/reference/android/provider/package-summary.html



Content Providers

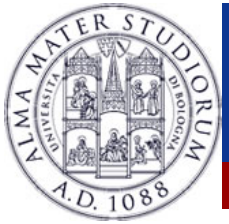




Example: contacts

- Query the contacts content provider
- Contacts information are shared among applications
- You need to request a permission

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```



Contacts: code

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Cursor cursor =
       getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
       null, null, null, null);
    while (cursor.moveToNext()) {
       String contactName = cursor.getString(cursor.getColumnIndex(
        ContactsContract.Contacts.DISPLAY_NAME));
    }
    cursor.close();
}
```