Databases and Content Providers

Dominic Duggan
Stevens Institute of Technology

Parcels

- Parcelable == Serializable for Android
  - Do not use Serializable in Android apps!
- Use: Fast IPC

```
public class Bundle {
   public int getInteger(String key);
   public void putInteger(String key, int value);
   public <T extends Parcelable>
      T getParcelable(String key);
   public void
      putParcelable(String key, Parcelable value);
}
```

Parcels

```
public class Book implements Parcelable {
  public String title;
  public float price;

  public Book(Parcel in) {
    this.title = in.getString();
    this.price = in.getFloat();
  }

  public void writeToParcel(Parcel out) {
    out.writeString(this.title);
    out.writeFloat(this.price);
  }
  public int describeContents() {
    return 0;
  }
}
```

3

Parcels

```
public class Book extends Parcelable {
  public String title;
  public float price;

  public static Creator<Book> CREATOR =
    new Creator<Book>() {
      public Book createFromParcel(Parcel in) {
        return new Book(in);
      }
      public Book[] newArray(int size) {
        return new Book[size];
      }
   }
}
```

4

Cross-App IPC

- Send data as a Bundle
- Receiving app may need to set classloader:

```
Message message;
Bundle data = message.getData();
data.setClassLoader(Book.class.getClassLoader());
Book book = data.getParcelable(BOOK_KEY);
```

5

Saving and Retrieving Data

- Simple Application Data
- Files
- SQLite Databases
- Content Providers

6

SIMPLE APPLICATION DATA

Simple Application Data

- Save simple data as name-value pairs
- Two scenarios:
  - User preferences (`SharedPreferences`)
  - Application UI state between calls to subactivities
    - `onPause()`
    - `onSaveInstanceState (Bundle outState)`
    - `onDestroy()`
    - `onCreate(Bundle inState)`
    - `onRestoreInstanceState(Bundle inState)`
    - `onStart()`
    - `onResume()`

User Preferences

```
public static final String MY_PREFS = "mySharedPreferences";
public static final String USERNAME = “username”;
public static final String PASSWORD = “password”;

protected void savePreferences(String chatUsername, char[] password) {
  // Create or retrieve the shared preference object.
  int mode = Activity.MODE_PRIVATE;
  SharedPreferences mySharedPreferences =
              getSharedPreferences(MY_PREFS, mode);

  // Retrieve an editor to modify the shared preferences.
  SharedPreferences.Editor editor = mySharedPreferences.edit();

  // Store new primitive types in the shared preferences object.
  editor.putString(USERNAME, chatUsername);
  editor.putString(PASSWORD, new String(password));

  // Commit the changes.
  editor.commit();
}
```

9

User Preferences

```
public static final String MY_PREFS = "mySharedPreferences";
public static final String USERNAME = “username”;
public static final String PASSWORD = “password”;

public void loadPreferences() {
  // Restore preferences
  int mode = Activity.MODE_PRIVATE;
  SharedPreferences mySharedPreferences =
              getSharedPreferences(MY_PREFS, mode);

  String username = mySharedPreferences.getString(USERNAME, null);

  char[] password =
              mySharedPreferences.getString(PASSWORD, "").toCharArray();
}
```

10

User Preferences

- Preferences
  - Cached in-memory prefs object
  - Shared by app components
  - Saved in app file space
- Concurrency and reliability
  - Saving is atomic
  - No concurrency control i.e. no locking
  - No transactions i.e. keep prefs file small

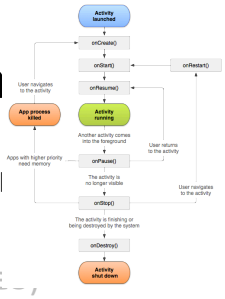
11

User Preferences

- Classes
  - `Preferences`: just one file
  - `SharedPreferences`: multiple files
  - `PreferenceActivity`: UI for setting preferences
- Modes
  - `MODE_PRIVATE`: only visible to app
  - `MODE_MULTI_PROCESS`: for multi-process app
    - Always check file for updates

12

Simple Application Da

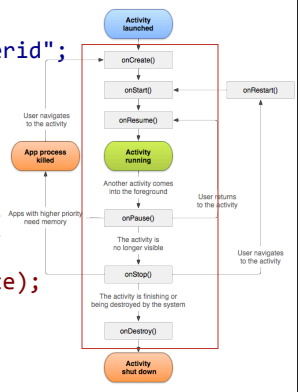


- Save simple data as name-value pai
- Two scenarios:
  - User preferences (SharedPreference
  - Application UI state between calls to subactivities
    - `onPause()`
    - `onSaveInstanceState (Bundle outState)`
    - `onDestroy()`
    - `onCreate(Bundle inState)`
    - ~~`onRestoreInstanceState(Bundle inState)`~~
    - `onStart()`
    - `onResume()`

13

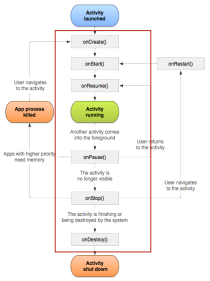
Application UI State

```
// Save UI state while activity is not active
// (i.e. UI state for a single user session)

private static final String USERID_KEY = ”userid";

private String loggedInUser;

@Override
public void onSaveInstanceState(
                  Bundle savedInstanceState) {

  super.onSaveInstanceState(savedInstanceState);

  savedInstanceState.putString(
              USERID_KEY,
              loggedInUser);
}
```



14

Application UI State




```
// Save UI state while activity is not active
// (i.e. UI state for a single user session)

private static final String USERID_KEY = ”userid";

private String loggedInUser;

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   loggedInUser = savedInstanceState.getString(USERID_KEY);
}
```

15

FILES

16

Saving and Loading Files

- Standard Java I/O
- `openFileInput` and `openFileOutput` for private files on internal storage:

```
String FILE_NAME = "tempfile.tmp";
FileOutputStream fos =
     openFileOutput(FILE_NAME,
                    Context.MODE_PRIVATE);
FileInputStream fis = openFileInput(FILE_NAME);
```

- Files can only be read/written in the current application folder

17

Saving and Loading Files

- Standard Java I/O
- `getExternalStoragePublicDirectory` for public files on external storage:

```
File album = new File(
   Environment.getExternalStoragePublicDirectory(
      Environment.DIRECTORY_PICTURES),
   albumName);
FileInputStream fis = new FileInputStream(album);
```

- Files stored on the SD card, shared with other apps

18

Saving and Loading Files

- Standard Java I/O
- `getExternalFilesDir` for private files on external storage:

```
File file = new File(
   getExternalFilesDir(null),
   fileName);
FileInputStream fis = new FileInputStream(file);
```

- Files stored on the SD card (protections may be circumvented by user)

19

Files

- External file resources res/raw folder

```
Resources myResources = getResources();
InputStream myFile =
   myResources.openRawResource(R.raw.myfilename);
```

- Application Context: tools for file management
  - `deleteFile`: remove internal files created by the current application
  - `fileList`: return String array of internal files created by the current application

20

SQLITE DATABASE

SQLite Databases

- Open source, lightweight, single-tier relational DBMS: http://www.sqlite.org
  - Used in iPhone and iPod Touch
- ContentValues: used to insert new rows into database tables
- Cursors: results of queries, with methods:

```
moveToFirst
moveToNext
getCount
moveToPosition
getPosition
etc
```

SQLiteOpenHelper

- Best practice patterns for creating, opening, upgrading databases

```
 private static class myDbHelper extends SQLiteOpenHelper {

   public myDbHelper(Context context, String name,
                     CursorFactory factory, int version) {
      super(context, name, factory, version);
   }

   // Called when no database exists in disk
   // and the helper class needs to create a new one.
   public void onCreate(SQLiteDatabase _db) {
      _db.execSQL(DATABASE_CREATE);
   }
}
```


SQLiteOpenHelper

```
private static class myDbHelper extends SQLiteOpenHelper {

   // Database version mismatch: version on disk
   // needs to be upgraded to the current version.
   public void onUpgrade(SQLiteDatabase _db, int _oldVersion,
                                             int _newVersion) {
      // Log the version upgrade.
      Log.w("TaskDBAdapter",
            "Upgrading from version " +  _oldVersion
                           + " to " +  _newVersion);

      // Upgrade: drop the old table and create a new one.
      _db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);

      // Create a new one.
      onCreate(_db);
   }
}
```

Working with Android Databases

- Create a database adaptor to encapsulate database interactions
  - Strongly typed methods
    - adding, removing, updating items
  - Handle queries
  - Wrap creating, opening, upgrading databases
  - Define static database constants

25

Example Database Adapter I

```
public class BookAdapter {

  private static final String DATABASE_NAME = ”books.db";
  private static final String DATABASE_TABLE = ”Books";
  private static final int DATABASE_VERSION = 1;

  // The index (key) column name for use in where clauses.
  public static final String _ID = "_id";

  // The name and column index of each column in your database.
  public static final String TITLE = ”title";
  public static final String AUTHOR = ”author";

  public static final int TITLE_KEY = 1;
  public static final int AUTHOR_KEY = TITLE_KEY + 1;

...
```

26

Example Database Adapter I

```
public class BookAdapter {

  // SQL Statement to create a new database.
  private static final String DATABASE_CREATE =
     “create table " + DATABASE_TABLE + " ("
        + _ID + " integer primary key, "
        + TITLE + " text not null, ”
        + AUTHOR + " text not null “
        + “)”;
```

27

Example Database Adapter I

```
public class BookAdapter {

  ...
  // Variable to hold the database instance
  private SQLiteDatabase db;

  // Context of the application using the database.
  private final Context context;

  // Database open/upgrade helper
  private myDbHelper dbHelper;

  public MyDBAdapter(Context _context) {
    context = _context;
    dbHelper = new myDbHelper(context, DATABASE_NAME, null,
                              DATABASE_VERSION);
  }
```

28

```
// Variable to hold the database instance
 private SQLiteDatabase db;

// Context of the application using the DB.
private final Context context;

// Database open/upgrade helper
private myDbHelper dbHelper;

public BookAdapter open() throws SQLException {
  db = dbHelper.getWritableDatabase();
  return this;
}

public void close() { db.close(); }

public long insertEntry(Book book) {
  ContentValues contentValues = new ContentValues();
  contentValues.putString(BOOK_TITLE, book.title);  // etc
  return db.insert(DATABASE_TABLE, null, contentValues);
}

public boolean removeEntry(long bookId) {
  return db.delete(DATABASE_TABLE,
                   _ID + "=" + bookId, null) > 0;
}
```

29

```
// Variable to hold the database instance
 private SQLiteDatabase db;

// Context of the application using the DB.
private final Context context;

// Database open/upgrade helper
private myDbHelper dbHelper;

public Cursor getAllEntries ()
  return db.query(DATABASE_TABLE,
                  new String[] {_ID, TITLE, AUTHOR},
                  null, null, null, null, null);
}

public Book getEntry(long bookId) {
  Book book = new Book();
  // TODO Return cursor to row and populate instance of Book
  return book;
}

public int updateEntry(long bookId, Book book) {
  String where = _ID + " = " + bookId;
  ContentValues contentValues = new ContentValues();
  // TODO Fill in the ContentValue based on the new object
  return db.update(DATABASE_TABLE, contentValues, where, null);
}
```

30

```
// Variable to hold the database instance
 private SQLiteDatabase db;

// Context of the application using the DB.
private final Context context;

// Database open/upgrade helper
private myDbHelper dbHelper;
```

```
public Cursor getAllEntries () {
  String[] projection = {_ID, TITLE, AUTHOR};
  return db.query(DATABASE_TABLE,
                  projection,
                  null, null, null, null, null);
}

public Cursor getByName(String title) {
  String[] projection = {_ID, TITLE, AUTHOR};
  String selection = TITLE + "=" + title;
  return db.query(DATABASE_TABLE,
                  projection,
                  selection,
                  null, null, null);
}
```

31

```
public Cursor getByName(String title) {
  String[] projection = {_ID, TITLE, AUTHOR};
  String selection = TITLE + "=" + title;
  return db.query(DATABASE_TABLE,
                  projection,
                  selection,
                  null, null, null);
}

public Cursor getByName(String title) {
  String[] projection = {_ID, TITLE, AUTHOR};
  String selection = TITLE + "=?";
  String[] selectionArgs = { title };
  return db.query(DATABASE_TABLE,
                  projection,
                  selection,
                  selectionArgs,
                  null, null);
}
```

32

Cursors: Querying a Database

```
int BALANCE_COLUMN = 2;
MyDbAdapter myDBA;
Cursor c = myDBA.getAllEntries();
float total = 0f;
// Make sure there is at least one row.
if (c.moveToFirst()) {
   // Iterate over each cursor.
   do {
     float balance = c.getFloat(BALANCE_COLUMN);
     total += balance;
   } while(c.moveToNext());
}

Float averageBalance = total / c.getCount();
```

33

Inserting Rows

```
SQLiteDatabase db;
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();

// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
// ... Repeat for each column ...

// Insert the row into your table
db.insert(DATABASE_TABLE,
          null,
          newValues);
```

```
bookDBA.insertEntry (new Book(…));
```

34

Updating and Deleting Rows

```
// Define the updated row content.
ContentValues updatedValues = new ContentValues();
// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
// ... Repeat for each column ...

where = _ID + "=" + rowId;

// Update the row with the specified index with the new values.
db.update(DATABASE_TABLE,
          newValues,
          where,
          null);
```

```
bookDBA.updateEntry (bookId,
                     new Book(…));
```


Updating and Deleting Rows

```
// Define the updated row content.
ContentValues updatedValues = new ContentValues();
// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
//[ ... Repeat for each column ... ]

where = KEY_ID + "=" + rowId;

// Update the row with the specified index with the new values.
db.update(DATABASE_TABLE,
          newValues,
          where,
          null);
```

```
bookDBA.updateEntry (bookId,
                     new Book(…));
```

```
// Delete a row
db.delete(DATABASE_TABLE,
          where,
          null);
```

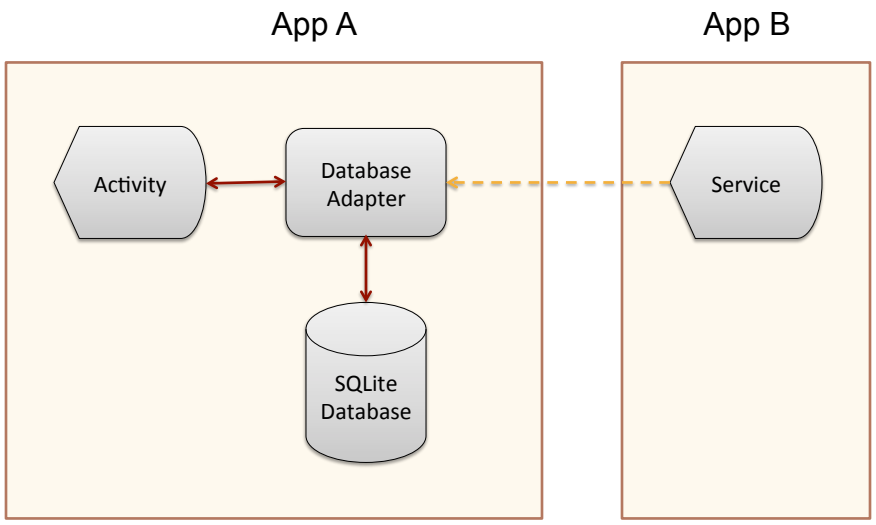
```
bookDBA.removeEntry (bookId);
```

CONTENT PROVIDERS

37

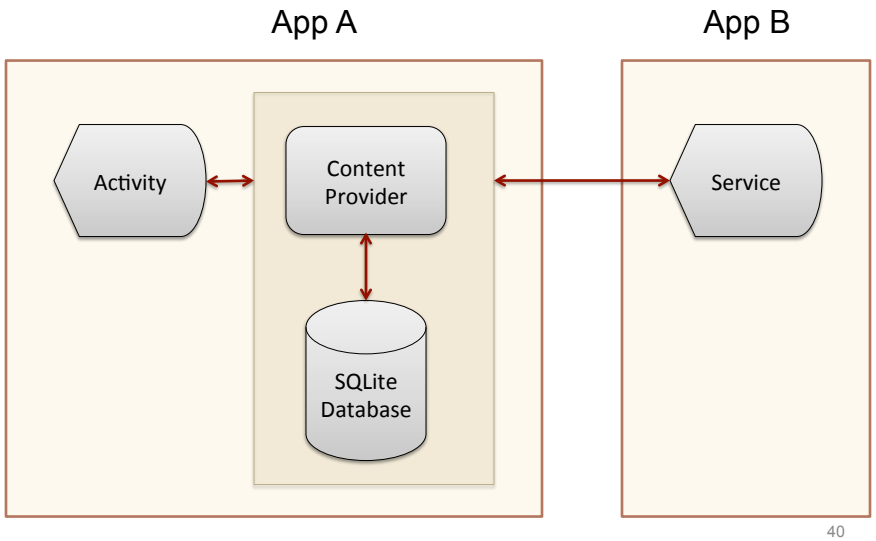
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android ...>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WIFI" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity android:name=".ChatServer" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <provider
            android:name=".MessageProvider"
            android:authorities="edu.stevens.cs522.chat.messages"
            android:exported="false" />

    </application>

</manifest>
```

Apps and Databases



39

Apps and Databases



40

Content Providers

- Generic interface for sharing data
- Content resolver
  - for obtaining content providers
- Content providers resolved using URIs
- Data returned as cursors
  - from content provider queries

41

Content Providers

```
ContentResolver cr = getContentResolver();

// Return all rows
Cursor allRows =
   getContentResolver().query(MyProvider.CONTENT_URI,
                              null, null, null, null);

// Return all cols for rows where col 3 equals set value
// and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor someRows =
   getContentResolver().query(MyProvider.CONTENT_URI,
                              null, where, null, order);
```

42

Content Providers: Insert

```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();

// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
// ... Repeat for each column ...

Uri myRowUri =
   getContentResolver().insert(MyProvider.CONTENT_URI,
                               newValues);
```

43

Content Providers: Insert

```
// Create a new row of values to insert.
ContentValues[] valueArray = new ContentValues[5];

// Create an array of new rows

int count =
   getContentResolver().bulkInsert(MyProvider.CONTENT_URI,
                                   valueArray);
```

44

Content Providers: Delete

```
// Remove a specific row.
getContentResolver().delete(myRowUri, null, null);
```

content://contacts/people/17

45

Content Providers: Delete

```
// Remove a specific row.
getContentResolver().delete(myRowUri, null, null);

// Remove the first five rows.
where = "_id < 5";
getContentResolver().delete(MyProvider.CONTENT_URI,
                            where, null);
```

content://contacts/people/

46

Content Providers: Update

```
// Create a new row of values to insert.
newValues = new ContentValues();

// Create a replacement map, specifying which columns to
// update, and what values to assign to each of them.
newValues.put(COLUMN_NAME, newValue);

// Apply to the first 5 rows.
where = "_id < 5";

getContentResolver().update(MyProvider.CONTENT_URI,
                            newValues, where, null);
```

47

Accessing Files in a Content Provider

```
// Insert a new row into your provider, returning its URI.
Uri uri =
   getContentResolver().insert(MyProvider.CONTENT_URI,
                               newValues);

try {
   // Open an output stream using the new row's URI.
   OutputStream outStream =
              getContentResolver().openOutputStream(uri);

   // Compress your bitmap and save it into your provider.
   sourceBitmap.compress(Bitmap.CompressFormat.JPEG,
                         50,
                         outStream);
} catch (FileNotFoundException e) { }
```

48

Native Android Content Providers

```
// Using the Media Store
android.provider
   .MediaStore.
   .Images
   .Media
   .insertImage(
      getContentResolver(),
      sourceBitmap,
      "my_cat_pic",
      "Photo of my cat!");
```

49

Using the Contacts Database

```
// Get a cursor over every contact.
Cursor cursor =
   getContentResolver().query(People.CONTENT_URI, null,
                              null, null, null);

// Let the activity manage the cursor lifecycle.
// DEPRECATED (see later)
startManagingCursor(cursor);

// Use convenience properties to get index of the columns
int nameIdx = cursor.getColumnIndexOrThrow(People.NAME);

int phoneIdx =
    cursor.getColumnIndexOrThrow(People.NUMBER);
```

50

Using the Contacts Database

```
// Get a cursor over every contact.
Cursor cursor =
   getContentResolver().query(People.CONTENT_URI, null,
                              null, null, null);
...

String[] result = new String[cursor.getCount()];

if (cursor.moveToFirst())
   do {
      // Extract the name and phone number.
      String name = cursor.getString(nameIdx);
      String phone = cursor.getString(phoneIdx);
      result[cursor.getPosition()] =
                 name + " (" + phone + ")";
   } while (cursor.moveToNext());
```

51