ANDROID

Data Storage
23.12.2015

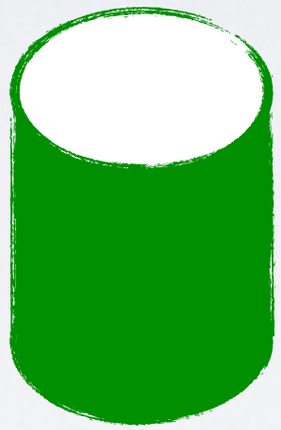
MOBILE APPLICATIONS PROGRAMMING

Krzysztof Pawłowski

Polsko-Japońska Akademia Technik Komputerowych

STORAGE OPTIONS

Shared Preferences



SQLite Database

Internal Storage

External Storage

SHARED PREFERENCES

- key-value pairs of primitive data types
- primitive data: booleans, floats, ints, longs, and strings
- data will persist across user sessions (even if your application is killed)

SHARED PREFERENCES EXAMPLE

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";
    @Override

    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .
       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       SharedPreferences.Editor editor = settings.edit();
       editor.putBoolean("silentMode", mSilentMode);

       editor.commit(); // Commit the edits!
    }
}
```

INTERNAL STORAGE

- device's internal storage
- files private to application
- removed when application uninstalled

INTERNAL STORAGE

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

INTERNAL STORAGE - USEFUL METHODS

- getCacheDir() - opens the internal directory where your application should save temporary cache files

- getDir() - creates (or opens an existing) directory within your internal storage space

INTERNAL STORAGE - USEFUL METHODS

- deleteFile() - deletes a file saved on the internal storage

- fileList() - returns an array of files currently saved by your application

EXTERNAL STORAGE

- removable storage media (such as an SD card)
- can become unavailable if the user mounts the external storage on a computer or removes the media
- all applications can read/write/remove files placed on the external storage

EXTERNAL STORAGE - PERMISSIONS

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

EXTERNAL STORAGE - PERMISSIONS

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

EXTERNAL STORAGE - CHECKING AVAILABILITY

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

EXTERNAL STORAGE - GETTING A DIRECTORY

- Public directories:
  - Pictures/
  - Music/
  - Ringtones/
- Environment.getExternalStoragePublicDirectory()

EXTERNAL STORAGE - GETTING A DIRECTORY EXAMPLE

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

SQLITE DATABASE

- serverless
- zero-configuration
- transactional
- sql based

SQLITE DATABASE - CONNECTION FROM ACTIVITY

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

SQLITE DATABASE - CONNECTION FROM ACTIVITY

- To get SQLiteDatabase object from class extending SQLiteOpenHelper we call:
  - getReadableDatabase() - to get db for read access
  - getWritableDatabase() - to get db for write access

SQLITEDATABASE CLASS

- beginTransaction()
- endTransaction()
- insert()
- update()
- delete()
- query()
- rawQuery()

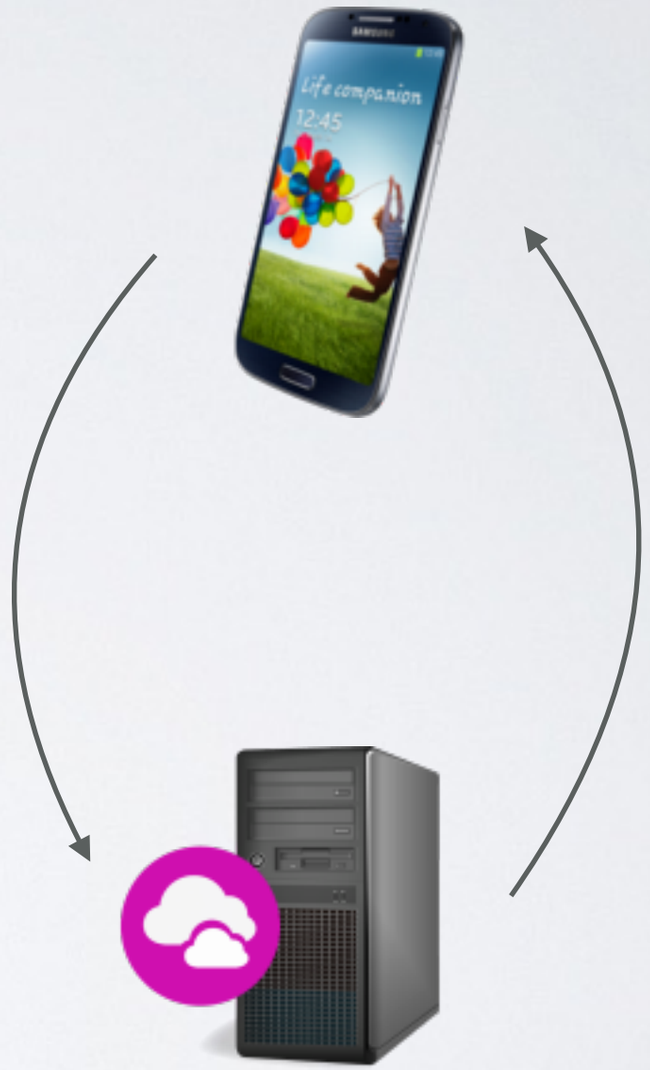
SQLITE - DEBUGGING ON DEVICE / EMULATOR

```
$ adb -s <emulator name> shell
# sqlite3 /data/data/<app_package>/databases/<db_name>.db
SQLite version 3.3.12
Enter ".help" for instructions
.... enter commands, then quit...
sqlite> .exit
```

DATA BACKUP

- allows copying persistent application data to remote "cloud" storage

- restore point for the application data and setting



DECLARING THE BACKUP AGENT

```
<manifest ... >
    ...
    <application android:label="MyApplication"
                 android:backupAgent="MyBackupAgent">
        <activity ... >
            ...
        </activity>
    </application>
</manifest>
```

REGISTERING FOR ANDROID BACKUP SERVICE

- To get your backup service key go to: https://developer.android.com/google/backup/signup.html

- Then paste your key into manifest file:

```
<application android:label="MyApplication"
             android:backupAgent="MyBackupAgent">
    ...
    <meta-data android:name="com.google.android.backup.api_key"
        android:value="AEdPqrEAAAAIDaYEVgU6DJnyJdBmU7KLH3kszDXLv_4DIsEIyQ" />
</application>
```

EXTENDING BACKUP AGENT

- Extend BackupAgent class if you need to:
  - version your data format
  - specify the portions of data the should be backed up and how each portion is then restored to the device
  - back up data in a database
- Otherwise use BackupAgentHelper

EXTENDING BACKUP AGENT

- When extending BackupAgent you need to override:
  - onBackup()
  - onRestore()

PERFORMING BACKUP

```
public void onBackup (ParcelFileDescriptor oldState, BackupDataOutput data,
                      ParcelFileDescriptor newState) {
    // Get the oldState input stream
    FileInputStream instream = new FileInputStream(oldState.getFileDescriptor());
    DataInputStream in = new DataInputStream(instream);
    try {
    // Get the last modified timestamp from the state file and data file
    long stateModified = in.readLong();
    long fileModified = mDataFile.lastModified();

    if (stateModified != fileModified) {
        // The file has been modified, so do a backup
        // Or the time on the device changed, so be safe and do a backup
    } else {
        // Don't back up because the file hasn't changed
        return;
    }
} catch (IOException e) {
    // Unable to read state file... be safe and do a backup
}
```

PERFORMING BACKUP

```
// Create buffer stream and data output stream for our data
ByteArrayOutputStream bufStream = new    ByteArrayOutputStream();
DataOutputStream outWriter = new    DataOutputStream(bufStream);
// Write structured data
outWriter.writeUTF(mPlayerName);
outWriter.writeInt(mPlayerScore);
// Send the data to the Backup Manager via the BackupDataOutput
byte[] buffer = bufStream.toByteArray();
int len = buffer.length;
data.writeEntityHeader(TOPSCORE_BACKUP_KEY, len);
data.writeEntityData(buffer, len);
```

PERFORMING BACKUP

```
FileOutputStream outstream = new FileOutputStream(newState.getFileDescriptor());
DataOutputStream out = new DataOutputStream(outstream);

long modified = mDataFile.lastModified();
    out.writeLong(modified);
}
```

More about performing backup:

https://developer.android.com/guide/topics/data/backup.html#PerformingBackup

PERFORMING RESTORE

```
@Override
public void onRestore(BackupDataInput data, int appVersionCode,
                      ParcelFileDescriptor newState) throws IOException {
    // There should be only one entity, but the safest
    // way to consume it is using a while loop
    while (data.readNextHeader()) {
        String key = data.getKey();
        int dataSize = data.getDataSize();

    // If the key is ours (for saving top score). Note this key was used when
    // we wrote the backup entity header
    if (TOPSCORE_BACKUP_KEY.equals(key)) {
        // Create an input stream for the BackupDataInput
        byte[] dataBuf = new byte[dataSize];
        data.readEntityData(dataBuf, 0, dataSize);
        ByteArrayInputStream baStream = new ByteArrayInputStream(dataBuf);
        DataInputStream in = new DataInputStream(baStream);

        // Read the player name and score from the backup data
        mPlayerName = in.readUTF();
        mPlayerScore = in.readInt();

        // Record the score on the device (to a file or something)
        recordScore(mPlayerName, mPlayerScore);
    } else {
        // We don't know this entity key. Skip it. (Shouldn't happen.)
        data.skipEntityData();
    }
}
```

PERFORMING RESTORE

```
    // Finally, write to the state blob (newState) that describes the restored data
    FileOutputStream outstream = new FileOutputStream(newState.getFileDescriptor());
    DataOutputStream out = new DataOutputStream(outstream);
    out.writeUTF(mPlayerName);
    out.writeInt(mPlayerScore);
}
```

BACKUP AGENT HELPER

- use BackupAgentHelper if you want to back up complete files
- no need to implement onBackup() and onRestore()

BACKING UP SHARED PREFERENCES

```
public class MyPrefsBackupAgent extends BackupAgentHelper {
    // The name of the SharedPreferences file
    static final String PREFS = "user_preferences";

    // A key to uniquely identify the set of backup data
    static final String PREFS_BACKUP_KEY = "prefs";

    // Allocate a helper and add it to the backup agent
    @Override
    public void onCreate() {
        SharedPreferencesBackupHelper helper =
                new SharedPreferencesBackupHelper(this, PREFS);
        addHelper(PREFS_BACKUP_KEY, helper);
    }
}
```

BACKING UP SHARED PREFERENCES

- That's it - code from the previous slide will cover backups and restores

BACKING UP SHARED OTHER FILES

```
public class MyFileBackupAgent extends BackupAgentHelper {
    // The name of the file
    static final String TOP_SCORES = "scores";
    static final String PLAYER_STATS = "stats";

    // A key to uniquely identify the set of backup data
    static final String FILES_BACKUP_KEY = "myfiles";

    // Allocate a helper and add it to the backup agent
    @Override
    public void onCreate() {
        FileBackupHelper helper = new FileBackupHelper(this,
                TOP_SCORES, PLAYER_STATS);
        addHelper(FILES_BACKUP_KEY, helper);
    }
}
```

BACKING UP SHARED OTHER FILES - THREAD SAFE VERSION

```
//MyFileBackupAgent

// Object for intrinsic lock
static final Object sDataLock = new Object();

@Override
public void onBackup(ParcelFileDescriptor oldState, BackupDataOutput data,
        ParcelFileDescriptor newState) throws IOException {
    // Hold the lock while the FileBackupHelper performs backup
    synchronized (MyActivity.sDataLock) {
        super.onBackup(oldState, data, newState);
    }
}

@Override
public void onRestore(BackupDataInput data, int appVersionCode,
        ParcelFileDescriptor newState) throws IOException {
    // Hold the lock while the FileBackupHelper restores the file
    synchronized (MyActivity.sDataLock) {
        super.onRestore(data, appVersionCode, newState);
    }
}
```

CHECKING THE RESTORE DATA VERSION

```
PackageInfo info;
try {
    String name = getPackageName();
    info = getPackageManager().getPackageInfo(name,0);
} catch (NameNotFoundException nnfe) {
    info = null;
}

int version;
if (info != null) {
    version = info.versionCode;
}
```

REQUESTING BACKUP

- You can request a backup operation at any time by calling dataChanged()

- Backup Manager then calls your backup agent's onBackup() method at an opportune time in the future.

REQUESTING RESTORE

- During the normal life of your application, you shouldn't need to request a restore operation.

- You can manually request a restore operation by calling requestRestore().

- Then, Backup Manager calls your onRestore() implementation, passing the data from the current set of backup data.

TESTING THE BACKUP AGENT

1. Install your application on a suitable Android system image.

2. Ensure that backup is enabled:
   `adb shell bmgr enable true`

3. Open your application and initialize some data.

4. Initiate a backup operation:
   `adb shell bmgr run`

5. Uninstall your application:
   `adb uninstall your.package.name`

6. Re-install your application.

APP INSTALL LOCATION

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:installLocation="preferExternal"
    ... >
```

- If you do *not* declare this attribute, your application will be installed on the internal storage only and it cannot be moved to the external storage.

WHEN USE INTERNAL STORAGE FOR APPS

- Services
- Alarm Services
- Input Method Engines
- Live Wallpapers
- App Widgets
- Account Managers
- Sync Adapter
- Device Administrators
- Broadcast Receivers listening for "boot completed"

WHEN USE EXTERNAL STORAGE FOR APPS

- All not mentioned on the previous slide
- Large games

CREDITS

All the code snippets and pictures come from:

https://developer.android.com/guide