LFE Medieninformatik • Prof. Dr. Heinrich Hußmann (Dozent), Alexander De Luca, Gregor Broll, Max-Emanuel Maurer (supervisors)

Praktikum Entwicklung von Mediensystemen mit Android

Storing, Retrieving and Exposing Data

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Outline

- Introduction
- Lightweight Storing
- Files
- Databases
- Network
- Content Providers
- Exercise 3

Introduction

- All application data are private to an application
- Mechanisms to make data available for other applications
- Some simple/basic applications do not require information to be stored
- More elaborated software needs storage/retrieval functionality for different functionalities like:
  - Preserving an application's status (paused, first startup, etc.)
  - Saving user preferences (font size, sound on/off, etc.)
  - Working with complex data structures (calendars, maps, etc.)
  - …

Purpose & Resource

- Depending on the purpose of storing data, Android offers approaches with different complexity:
  - Store and retrieve simple name/value pairs
  - File operations (read, write, create, delete, etc.)
  - SQLite databases to work with complex data structures
  - Network operations to store and retrieve data from a network
  - Content providers to read/write data from an application's private data

Preferences

- Application preferences are simple **name/value pairs** like "greeting=hello name" or "sound = off"
- To work with preferences, Android offers an extremely simple approach
- Preferences can only be shared with other components in **the same** package
- Preferences cannot be shared across packages
- Private preferences will not be shared at all

sound: off

font_size: 10pt

username: hugo

pem: rocks

Using Preferences

- **Reading** Preferences
  - Context.getSharedPreferences(String name, int mode) opens a set of preferences defined by “name”
  - If a name is assigned, the preferences set will be shared amongst the components of the same package
  - Activity.getPreferences(int mode) can be used to open a set that is private to the calling activity

Opens a preferences set with the name “Preferences” in private mode

```
SharedPreferences settings = getSharedPreferences("Preferences", MODE_PRIVATE);
boolean sound = settings.getBoolean("sound", false);
```

Reads a boolean parameter from the set. If the parameter does not exist, it will be created with the value defined in the second attribute. (other functions: getAll(), getInt(), getString(), etc.)

Using Preferences

- **Writing** Preferences
  - Changes on preferences are done using an Editor (SharedPreferences.Editor) object
  - Each setting has one global Editor instance to administrate changes
  - Consequence: each change will be available to every activity working with that preferences set

Gets the Editor instance of the preferences set

```
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("sound", false);
// COMMIT!!
editor.commit();
```

Writes a boolean to a parameter

Attention: Changes are not drawn back to the settings before the commit is performed

Files

- Files can be used to store bigger amounts of data then using preferences
- Android offers functionality to read/write files
- Only local files can be accessed

- **Advantage**: can store huge amounts of data
- Disadvantage: file update or changing in the format might result in huge programming effort

Working with Files

- **Reading** from files
  - Context.openFileInput(String name) opens a FileInputStream of a private file associated with the application
  - Throws a FileNotFoundException if the file doesn't exist

Open the file "test2.txt" (can be any name)

```
FileInputStream in = this.openFileInput("test2.txt");
…
in.close();
```

Don't forget to close the InputStream at the end

Working with Files

- **Writing** files
  - Context.openFileOutput(String name, int mode) opens a FileOutputStream of a private file associated with the application
  - If the file does not exist, it will be created
  - FileOutputStreams can be opened in append mode, which means that new data will be added at the end of the file

Open the file "test2.txt" for writing (can be any name)

```
FileOutputStream out = this.openFileOutput("test2.txt", MODE_APPEND);
…
in.close();
```

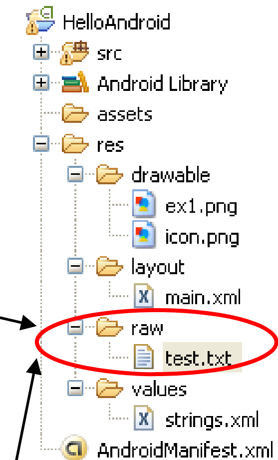
Using MODE-APPEND opens the file in append mode

Don't forget to close the InputStream at the end

Working with Files

Files

- **Reading static** files
  - ➢ To open static files packed in the application, use Resources.openRawResource (R.raw.mydatafile)
  - ➢ The files have to be put in the folder res/raw/



Get the contexts resources

```
InputStream in = this.getResources().openRawResource(R.raw.test);
…
in.close();
```

Don't forget to close the InputStream at the end

SQLite Databases

- In some cases, files are not efficient
  - If multi-threaded data access is relevant
  - If the application is dealing with complex data structures that might change
  - Etc.
- Therefore, Android comes with built-in SQLite support
- Databases are private to the package that created them
- Support for complex data types, e.g. contact information (first name, familiy name, address, ...)
- Databases should not be used to store files
- Hint: an example on how to use databases can be found in the SDK at samples/NotePad

SQLite Databases

- SQLite is a lightweight software library
- Implements a fully ACID-compliant database
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Size only several kilobytes
- Some SQL statements are only partially supported (e.g. ALTER TABLE)
- See http://www.sqlite.org/ for more information

Using Databases

SQLite Databases

- **Creating** a database
  - Context.createDatabase(String name, int version, int mode, CursorFactory factory) creates a new database and returns a SQLiteDatabase object
  - Throws a FileNotFoundException if the database could not be created

Create a database with the name "test.db" (can be any name)

```
SQLiteDatabase dbase = this.createDatabase("test.db",
1, MODE_PRIVATE, null);
```

Optional CursorFactory parameter

Using Databases

SQLite Databases

- **Deleting** a database
  - ➢ Context. deleteDatabase(String name) deletes the database with the specified name
  - ➢ Returns true if the database was successfully deleted or false if not (e.g. database does not exist)

Delete database "test.db"

```
boolean success = this.deleteDatabase("test.db);
```

Using Databases

SQLite Databases

- **Opening** a database
  - ➢ Context.openDatabase(String file, CursorFactory factory) opens an existing database and returns a SQLiteDatabase object
  - ➢ Throws a FileNotFoundException if the database does not exist yet

Create a database with the name “test.db” (can be any name)

```
SQLiteDatabase dbase = this.openDatabase("test.db", null);
…
dbase.close();
```

Optional CursorFactory parameter

Don’t forget to close the database at the end

Using Databases

SQLite Databases

- **Non-Query** SQL Statements
  - ➢ SQLiteDatabase.execSQL(String sql) can be used to execute non-query SQL statements, that is statements without a result
  - ➢ Includes CREATE TABLE, DROP TABLE, INSERT etc.
  - ➢ Examples:

Create a table with the name "test" and two parameters

```
dbase.execSQL("CREATE TABLE test (_id INTEGER PRIMARY KEY, someNumber
INTEGER);");
```

Insert a tuple into the database

```
dbase.execSQL("Insert into test (_id, someNumber) values(1,8);");
```

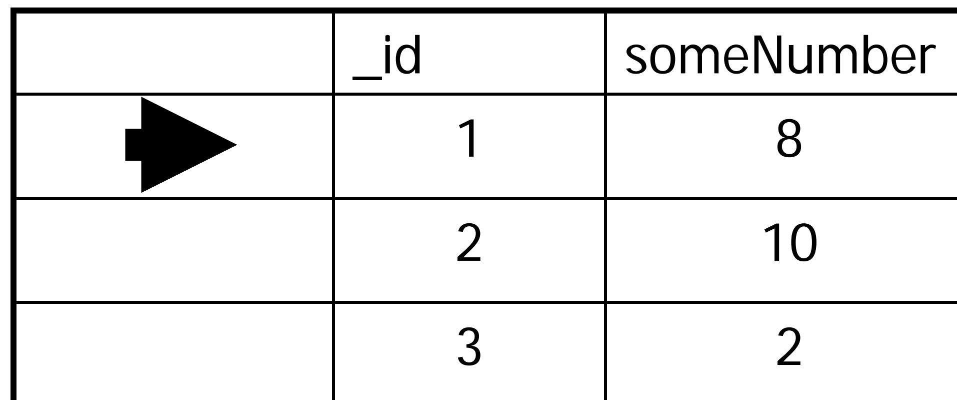
Drop the table "test"

```
dbase.execSQL("DROP TABLE test");
```

Using Databases

- **Query** SQL Statements - Cursors
  - Android uses cursors to navigate through query results
  - Cursors are represented by the object android.database.Cursor
  - A cursor is simply a pointer that “jumps” from one tuple of the query’s result to the next (or the previous or the first or …)
  - The cursor returns the data of the tuple it is located at the moment

Table “test”



	_id	someNumber
➡	1	8
	2	10
	3	2

Using Databases

SQLite Databases

To create a cursor, a query has to be executed either by SQL using rawQuery() or by more elaborated methods like query()

```
Cursor cur = dbase.rawQuery("SELECT * FROM test", null);

if (cur != null) {
    int numColumn = cur.getColumnIndex("someNumber");
    if (cur.first()) {
      do {
           int num = cur.getInt(numColumn);
           …do something with it…
         } while (cur.next());
    }
}
```

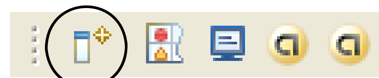
Attributes are retrieved with their index

Cursor offers different methods to retrieve different datatypes like getInt(int index) getString(int index) etc

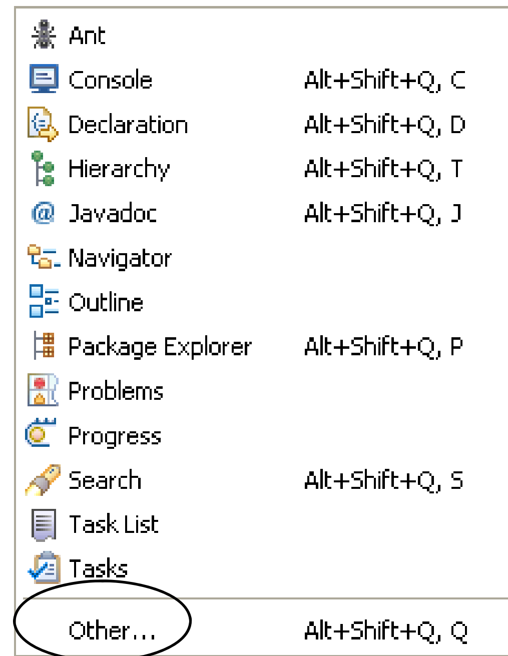
next() moves the cursor to the next row. It returns false if no more row is available. Other possible moves are previous() and first()

Using the IDE to Check Files and Databases

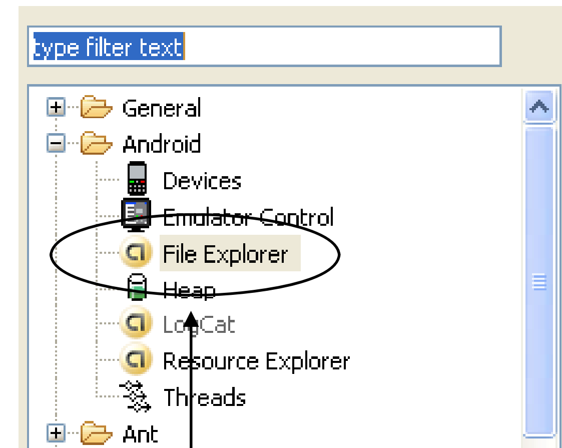
- The Android plug-in provides a view to check all created files and databases
- 1. Add File Explorer view to the IDE



a) click



b) click



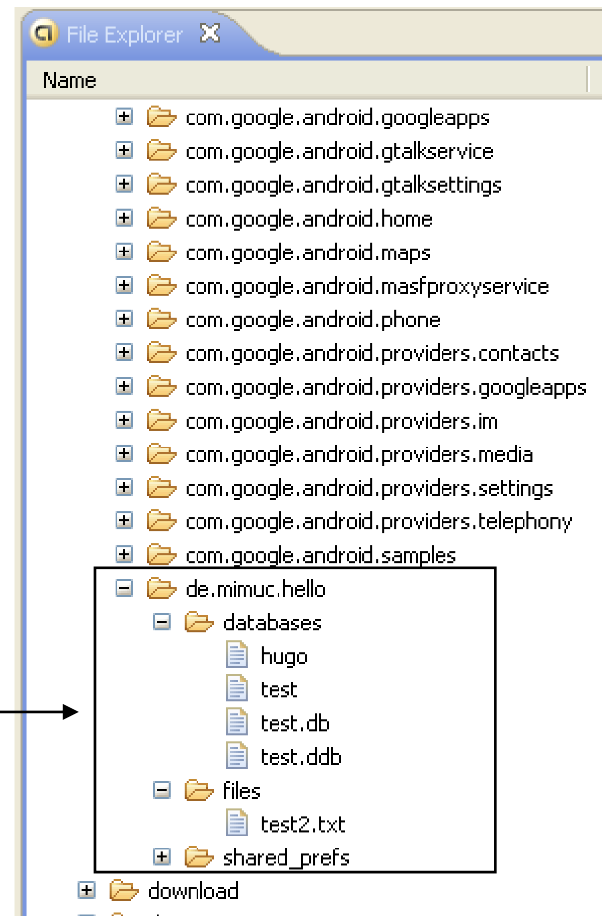
c) click

Using the IDE to Check Files and Databases

- 2. Check Files and Databases at /data/data/<package_name>/files|databases

click

The ultimate proof that Android accepts ANY file and database name



Network Access

- Android also supports network access to access files remotely (through the network)

- Two major packages:
  - java.net.* contains the standard Java network APIs
  - android.net.* adds additional helper classes to the standard Java APIs

Content Providers

- All preferences, files and databases created by an Android application are private
- To share data with other applications, an application has to **create** a Content Provider
- To retrieve data of another application its content provider has to be **called**
- Androids **native Content Providers** include:
  - CallLog: information about placed and received calls
  - Settings.System: system settings and preferences

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Exercise

- Chat-history Application
  - ➢Based on exercise 2
  - ➢Functionality
    - ▪ changing the status (available etc.) is stored and automatically set on starting the application
    - ▪ the chat history has to be stored automatically
      - o each message has to be stored together with a timestamp
    - ▪ two buttons to display the history
      - o of the day
      - o of all chat sessions
    - ▪ any storing mechanism is ok

•Any improvements on the design or additional functionality is encouraged

See you next meeting!