LMU
LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

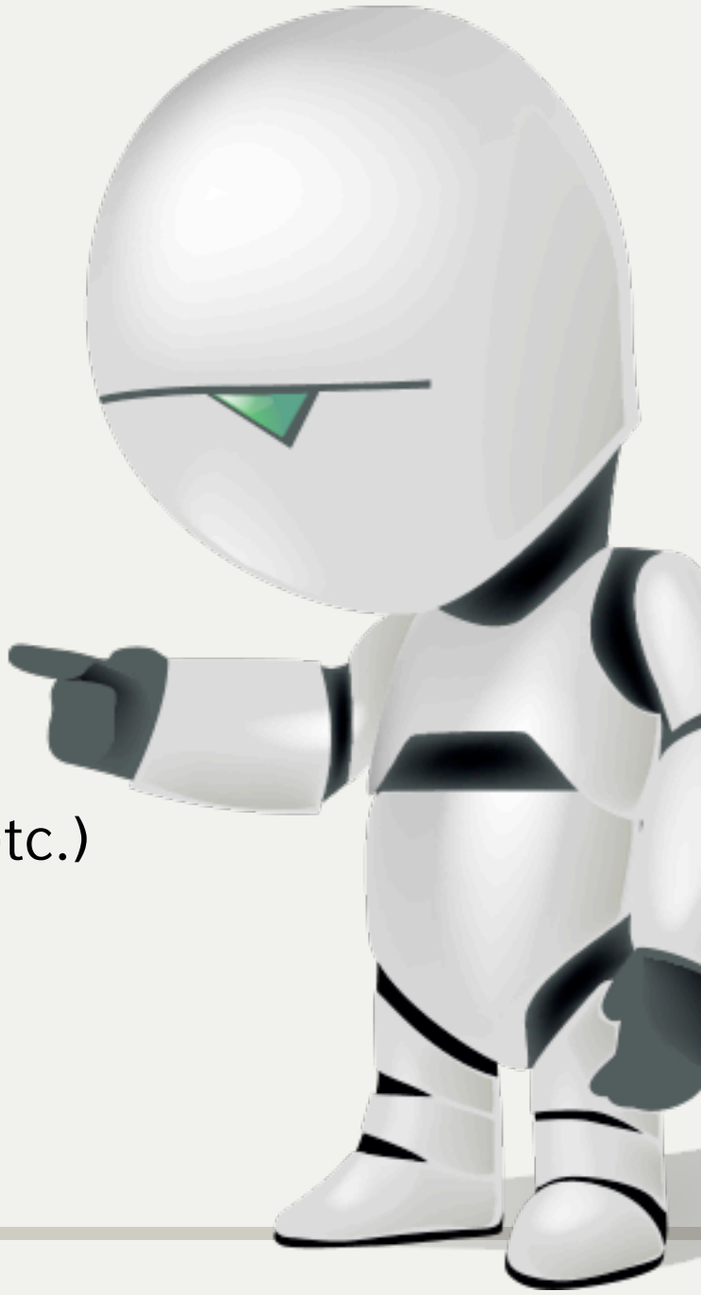
FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
ARBEITSGRUPPEN MEDIENINFORMATIK UND
MENSCH-MASCHINE-INTERAKTION

Praktikum Entwicklung Mediensysteme (für Master)

Storing, Retrieving and Exposing Data

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.)
  - …



Different Storage Methods

- 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

File-IO

SQLite-Databases

Network Storage

Content-Providers

Preferences

File-IO

SQLite-Databases

Network Storage

Content-Providers

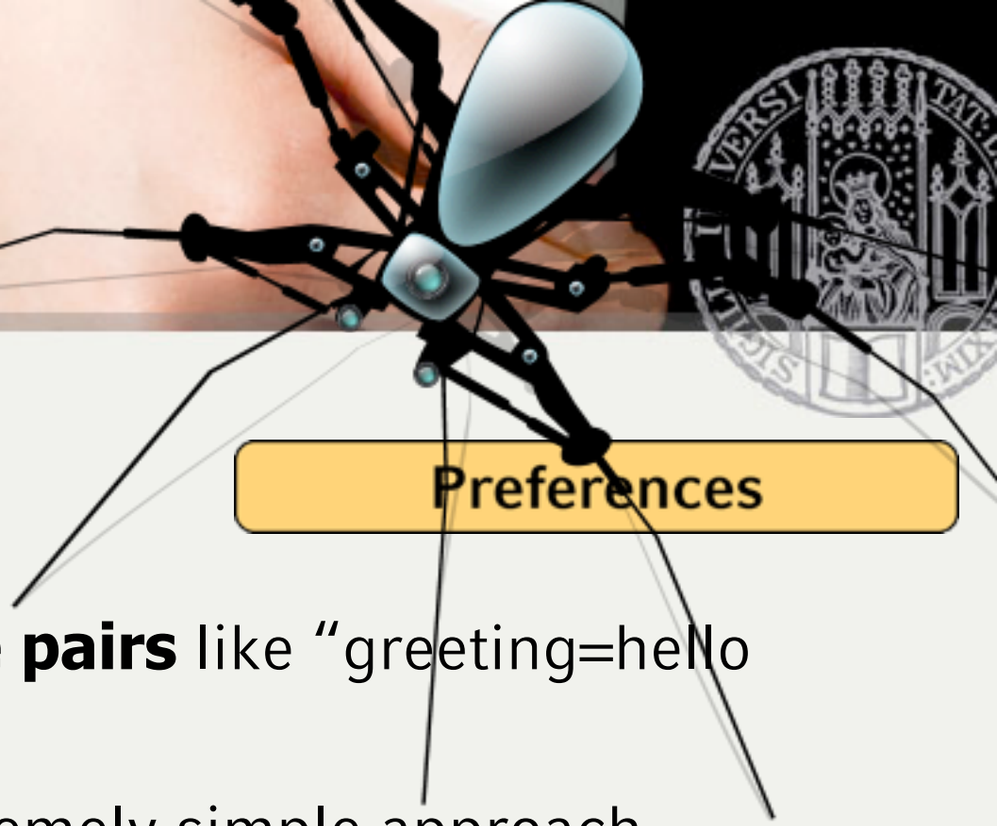
Preferences

File-IO

SQLite-Databases

Network Storage

Content-Providers



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
- Storage location is not defined and inaccessible for other applications

sound: off

font_size: 10pt

username: hugo

pem: rocks

Using Preferences

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

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

Preferences

File-IO

SQLite-Databases

Network Storage

Content-Providers

Files

File-IO

- Files can be used to store bigger amounts of data than 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

Reading Files

File-IO

- 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

Writing Files

File-IO

- 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();
```

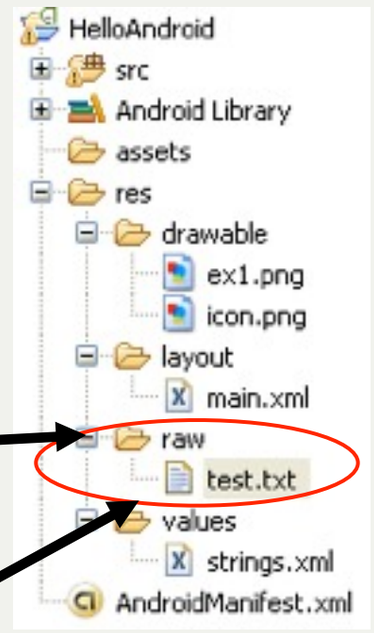
Using MODE-APPEND opens the file in append mode

Don’t forget to close the InputStream at the end

Static Files

File-IO

- 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

Using the SD-Card

File-IO

- Bigger amounts of data should usually be written/read from SD-Card
- Using the external storage requires permission
- Set it in Manifest.xml-File

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

Preferences

File-IO

SQLite-Databases

Network Storage

Content-Providers

SQLite Databases

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

SQLite Databases

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)
- Only few types of data
- See http://www.sqlite.org/ for more information

Creating a Database

SQLite-Databases

- Opening a database should create it when needed
- Creating a database always means taking care of future Versions
- Version-Numbers make sure which kind of DB is currently used
- An extra class usually called „DBAdapter.java“ is used for all database access

SQLite-Databases

```
public class DBAdapter extends SQLiteOpenHelper {
    public static final String KEY_ROWID = "_id";
    private static final String TAG = "DBAdapter";

    private static final String DATABASE_NAME = "mydb";
    private static final String DATABASE_TABLE = "table_one";
    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_CREATE = "create table "+DATABASE_TABLE+" (" +
        KEY_ROWID + " integer primary key autoincrement);";

    private SQLiteDatabase db;

    public DBAdapter(Context ctx) {
        super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
        db=getWritableDatabase();
    }

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

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
        onCreate(db);
    }

}
```

Fetching Data

SQLite-Databases

- Data is provided using Cursors
- Cursors are the result of a specific query to the database holding the request result
- Cursors are traversed line by line
  - Similar to an Iterator in Java
- DBAdapter should provide request-methods that return such a Cursor

	id	someNumb
→	1	8
	2	10
	3	2

Fetching Data

SQLite-Databases

```
public Cursor getAllEntrys() {
    return db.query(DATABASE_TABLE, new String[] { KEY_ROWID }, null, null,
            null, null, null);
}
```

- Parameters
  - table: The table to query from
  - columns: Which columns to fetch
  - selection: the „Where“-Clause with placeholders?
  - selectionArgs: Values to fill placeholders
  - groupBy: SQL groupBy-Values
  - having: SQL having-Values
  - orderBy: How to order the resulting datasets

Insert, Update

SQLite-Databases

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

- Some examples:

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

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

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

SQLiteQueryBuilder

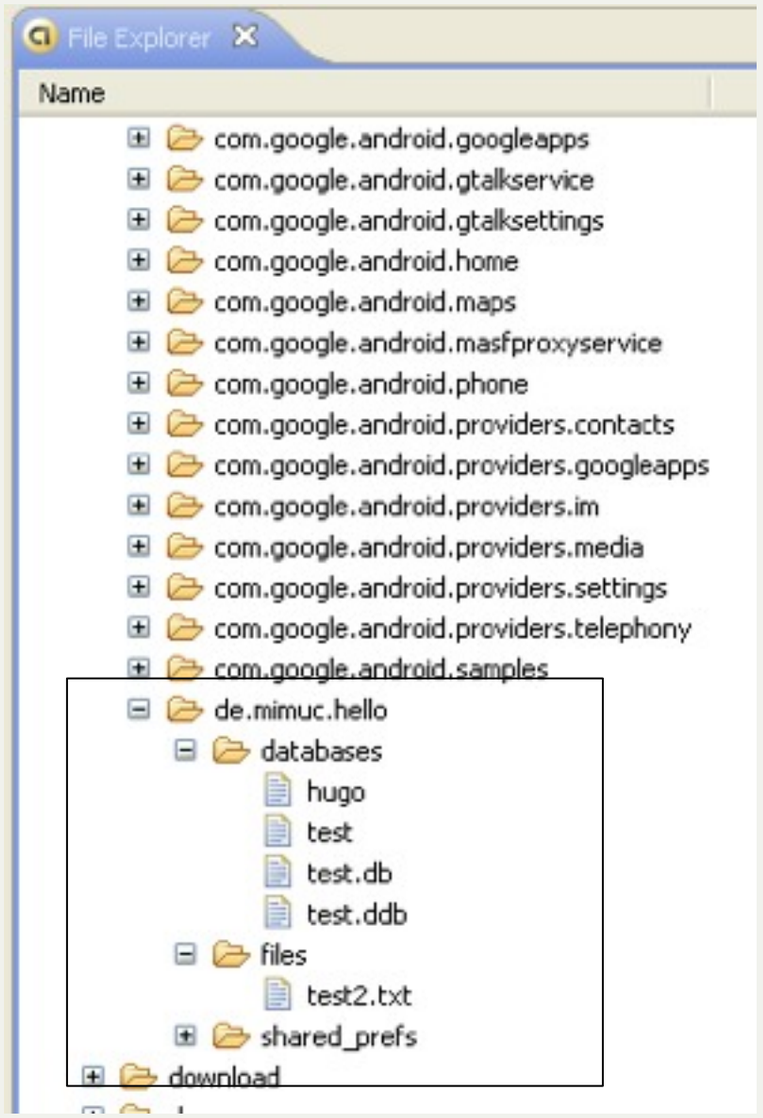
SQLite-Databases

- Optional interface to build correct SQL statements using code
- Usage:
  - Create new SQLLiteQueryBuilder object
  - Then use setTables, appendWhere, appendColumns
  - In the end, use query or buildQuery

Using the IDE to Check Files and Databases

SQLite-Databases

- FileExplorer-View
- Check Files and Databases at / data/data/<package_name>/ files|databases
- Only possible on a „rooted“ device/emulators.
- **Don‘t root the test devices!**



Preferences

File-IO

SQLite-Databases

Network Storage

Content-Providers

Network Access

Network Storage

- 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

Preferences

File-IO

SQLite-Databases

Network Storage

Content-Providers

Content Providers

Content-Providers

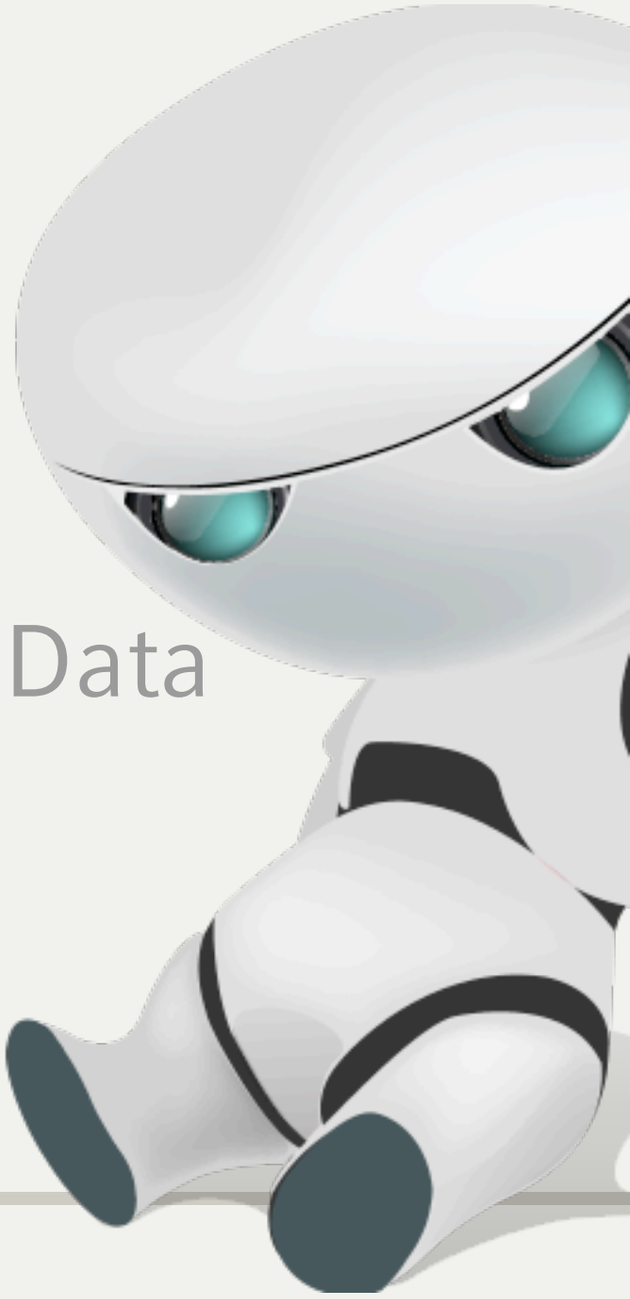
- All preferences, files and databases created by an Android application are private
- To share data with other applications, an application has to **create** a Content Provider
- To retrieve data of another application its content provider has to be **called**
- Androids **native Content Providers**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

LMU
LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
ARBEITSGRUPPEN MEDIENINFORMATIK UND
MENSCH-MASCHINE-INTERAKTION

Exercise 3

Storing, Retrieving and Exposing Data



Exercise 3


http://www.medien.ifi.lmu.de
Öffnen
Suche
STARTSEITE
STUDIENGANG
LEHRE
FORSCHUNG
PERSONEN
KONTAKT
INTERN
Arbeitsgruppen Medieninformatik und Mensch-Maschine-Interaktion
Verlauf anzeigen


17:50
Verlauf
http://www.medien.ifi.lmu.de
http://www.amazon.de

Exercise 3

- Fortführung der bisherigen Aufgabe
- In neues Projekt kopieren
- Datenbankbasierte Browser History erstellen
- History wird automatisch gefüllt (keine Duplikate)
- Kann über Anwendungsmenü geöffnet werden
- Zugriff auf alte Seiten der History möglich

Fragen?
Viel Spaß!