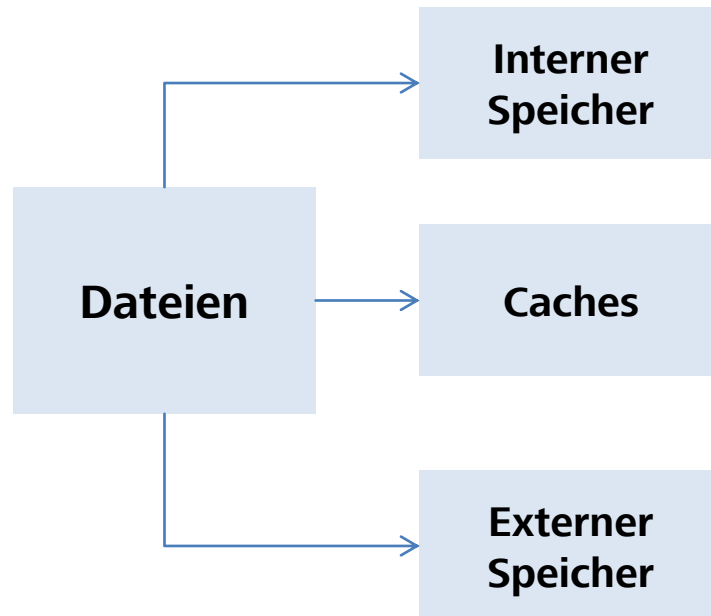


# Smart Home Control Android Persistenz

Prof.Dr.Ing.Birgit Wendholt

# Überblick

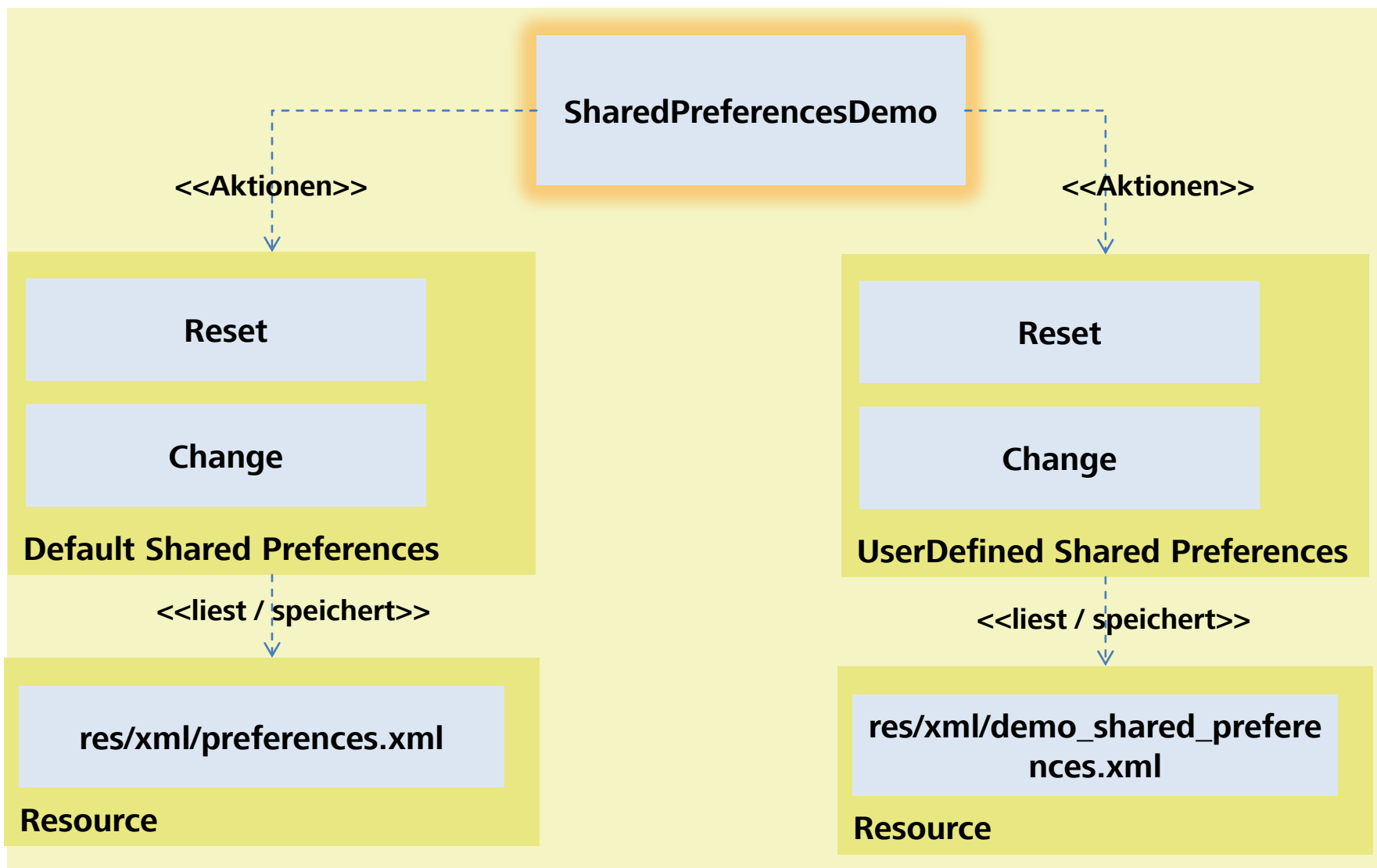


**Key-Value Paare**  
**SharedPreferences**

**SQL Datenbanken**  
**s**

# Speichern von Key-Value Paaren SharedPreferences

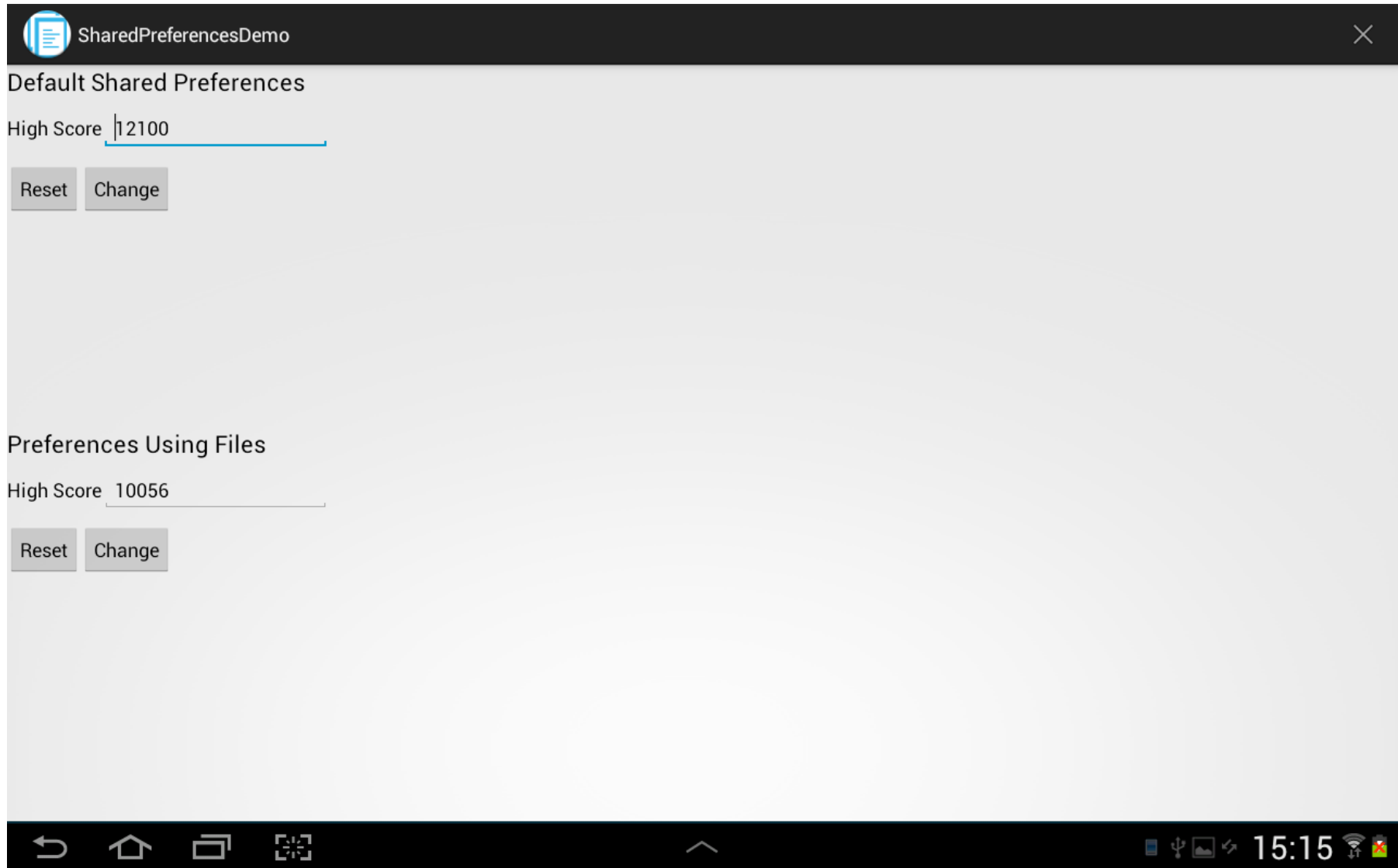
# Überblick über das Beispiel



# SharedPreferences


## (Source → LPPersistenceSharedPreferences)

- Dateien, die Key-Value Paare enthalten
  - private oder öffentliche Dateien
- Zugriff auf *SharedPreferences* (erzeugt eine Datei, wenn noch nicht existent)
  - *getSharedPreferences(aSPFile, modus)*: für mehrere Dateien, die sich die Komponenten einer Applikation teilen, referenziert über den Namen *aSPFile*
  - *getPreferences(modus)*: eine Datei, die nur eine Aktivität verwendet – Default-Name, daher kein Parameter
- *modus*:
  - Context.MODE\_PRIVATE
  - Context.MODE\_WORLD\_READABLE
  - Context.MODE\_WORLD\_WRITABLE



# Anlegen von Dateien für Präferenzen

- Anlegen von XML-Dateien im Verzeichnis *res/xml*
  - Default: preferences.xml
  - User Defined: beliebiger Name (im Beispiel: *demo\_shared\_preferences.xml*)
- Konvention für die Dateien:
  - Android XML Datei mit Resource Type *Preference* (siehe letzte Vorlesung)



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

    <Preference
        android:defaultValue="@integer/saved_high_score_default"
        android:key="@string/saved_high_score"
        android:title="@string/high_score" />

</PreferenceScreen>
```

# Laden und Lesen der Präferenzen

## Default

```
pref = getPreferences(  
    Context.MODE_PRIVATE);
```

## User Defined

```
sharedPref = getSharedPreferences(  
    getString(  
        R.string.preference_file_key),  
    Context.MODE_PRIVATE);
```

*definiert als  
demo\_shared\_preferences*



```
private void showPref(SharedPreferences pref, TextView tv) {  
    int defValue = getResources().getInteger(R.integer.saved_high_score_default);  
    int highScore = pref.getInt(  
        getString(R.string.saved_high_score),  
        defValue);  
    tv.setText(String.valueOf(highScore));  
}
```

*Zugriffsmethoden für  
unterschiedliche Datentypen  
werden unterstützt*





# Ändern und Schreiben von Präferenzen

*Änderungen immer über den Editor für Präferenzen,  
der konkurrierenden Zugriff auf Präferenzen regelt.*

*Änderungen müssen mit commit abgeschlossen werden!*

```
private void changePref(SharedPreferences sharedPref,  
                        EditText ahighScoreField) {  
  
    Editor editor = sharedPref.edit();  
    editor.putInt(getString(R.string.saved_high_score),  
        Integer.parseInt(ahighScoreField.getText().toString()));  
    editor.commit();  
}
```



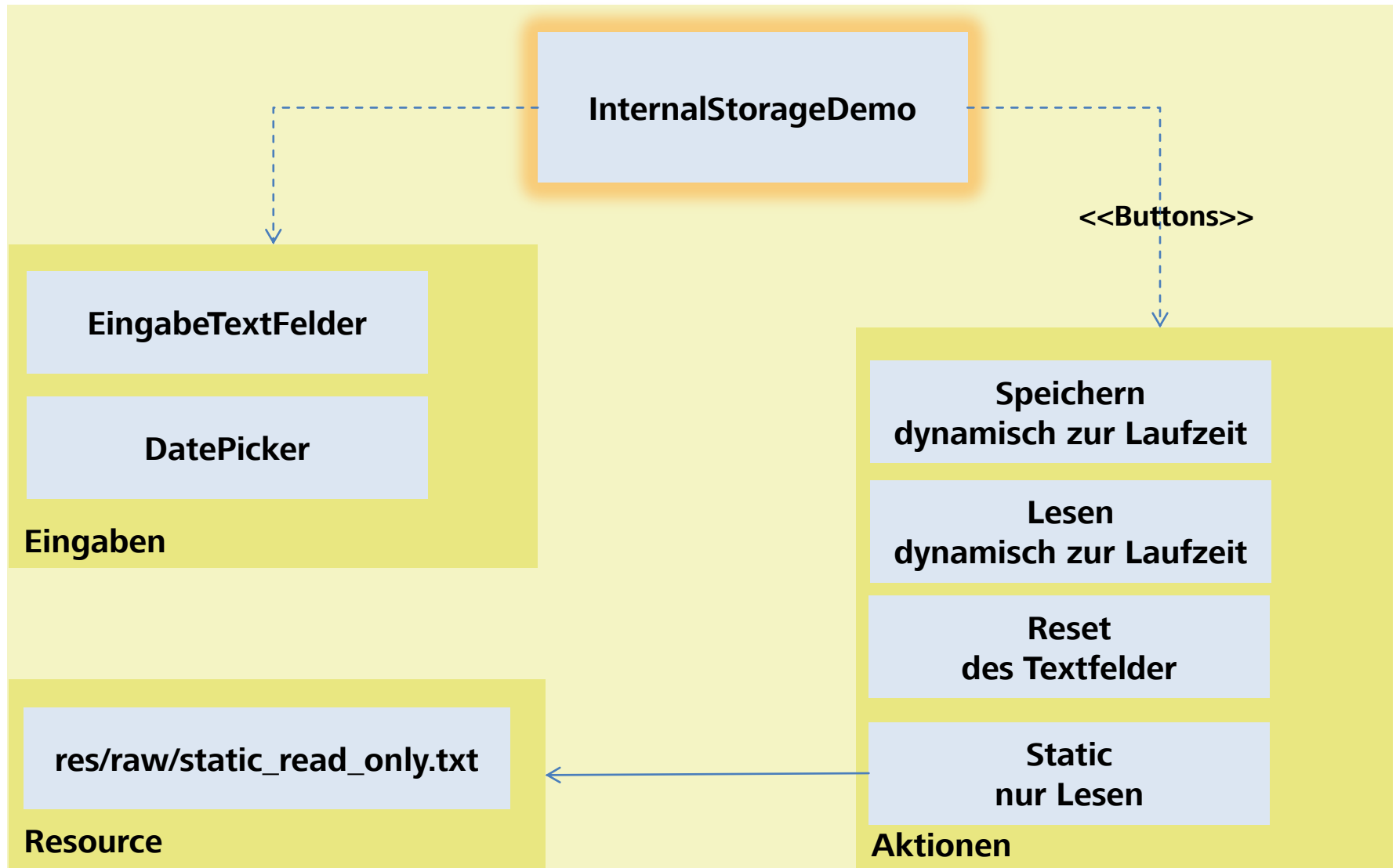
*Schreibmethoden für  
unterschiedliche Datentypen  
werden unterstützt.*


*Alle Änderungen sind persistent!*

# Dateien- Verwenden des internen Speichers

# Interner Speicher einer Applikation

- Privater Speicher einer Android Applikation
  - kein Zugriff durch andere Applikationen
  - kein Zugriff durch den Benutzer
  - Dateien werden gelöscht, wenn die Applikation deinstalliert wird.
- Varianten:
  - Erzeugen von Dateien zur Laufzeit
  - Verwenden einer statischen Datei zur Compilezeit




InternalStorageDemo

Dateiname

Bestellnummer

Bestellwert

Liefertermin

▲

03

▼

▲

Dez

▼

▲

2012

▼

04

Jan

2013

05

Feb

2014

Januar 2013

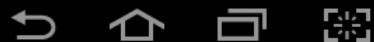
	S	M	D	M	D	F	S
1	30	31	1	2	3	4	5
2	6	7	8	9	10	11	12
3	13	14	15	16	17	18	19
4	20	21	22	23	24	25	26
5	27	28	29	30	31	1	2
6	3	4	5	6	7	8	9

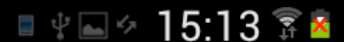
Speichern

Lesen

Reset

Lesen Def





Prof.Dr.B.Wendholt: Android Persistenz

13

# Schreiben und Lesen von Dateien im internen Speicher

## Schreiben

- Erzeugen: *openFileOutput(aFileName, modus)* → *FileOutputStream*
- *modus*:
  - Context.MODE\_PRIVATE
  - Context.MODE\_APPEND
  - Context.MODE\_WORLD\_READABLE
  - Context.MODE\_WORLD\_WRITABLE
- Wrappen mit BufferedWriter und Schreiben *write* und *newLine*
- Schließen des Streams mit *close*

## Lesen

- Öffnen mit *openFileInput(aFileName)* → *FileInputStream*.
- Wrappen mit BufferedReader und Lesen *readLine*
- Schließen des Stroms mit *close*

# Schreiben auf eine Datei des internen Speichers

```
// Lesen der Werte aus den Eingabe Feldern des GUI's
try {
    FileOutputStream fos = openFileOutput(file, Context.MODE_PRIVATE);
    BufferedWriter bfos = new BufferedWriter(new OutputStreamWriter(fos));
    bfos.write(bstNr); bfos.newLine();
    bfos.write(bstVal.toString()); bfos.newLine();
    bfos.write(String.valueOf(dayOfMonth) + ":");
    bfos.write(String.valueOf(month) + ":");
    bfos.write(String.valueOf(year)); bfos.newLine();
    bfos.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

# Lesen einer Datei des internen Speichers

```
String file = fileName.getText().toString();  
try {  
    FileInputStream fis = openFileInput(file);  
    BufferedReader bfis = new BufferedReader(new InputStreamReader(fis));  
    updateFieldsFrom(bfis);  
    bfis.close();  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```



# Aktualisieren der GUI Komponenten

```
private void updateFieldsFrom(BufferedReader bfis) throws IOException {  
  
    orderNumber.setText(bfis.readLine());  
    orderValue.setText(bfis.readLine());  
    String[] dateStrings = bfis.readLine().split(":");  
    int year = Integer.parseInt(dateStrings[2]);  
    int month = Integer.parseInt(dateStrings[1]);  
    int dayOfMonth = Integer.parseInt(dateStrings[0]);  
    datePicker.init(year, month, dayOfMonth, null);  
}
```

# Erzeugen einer statischen Datei zur Compilezeit

- Abgelegt im Verzeichnis *res/raw/*
- kann nur gelesen werden
- Öffnen der Datei
  - *getResources().openRawResource(R.raw.<filename\_id>)*  
→ *InputStream*
  - Wrappen mit einem *BufferedReader*

static\_read\_only.txt

bst0000000  
99.99  
02:01:2013

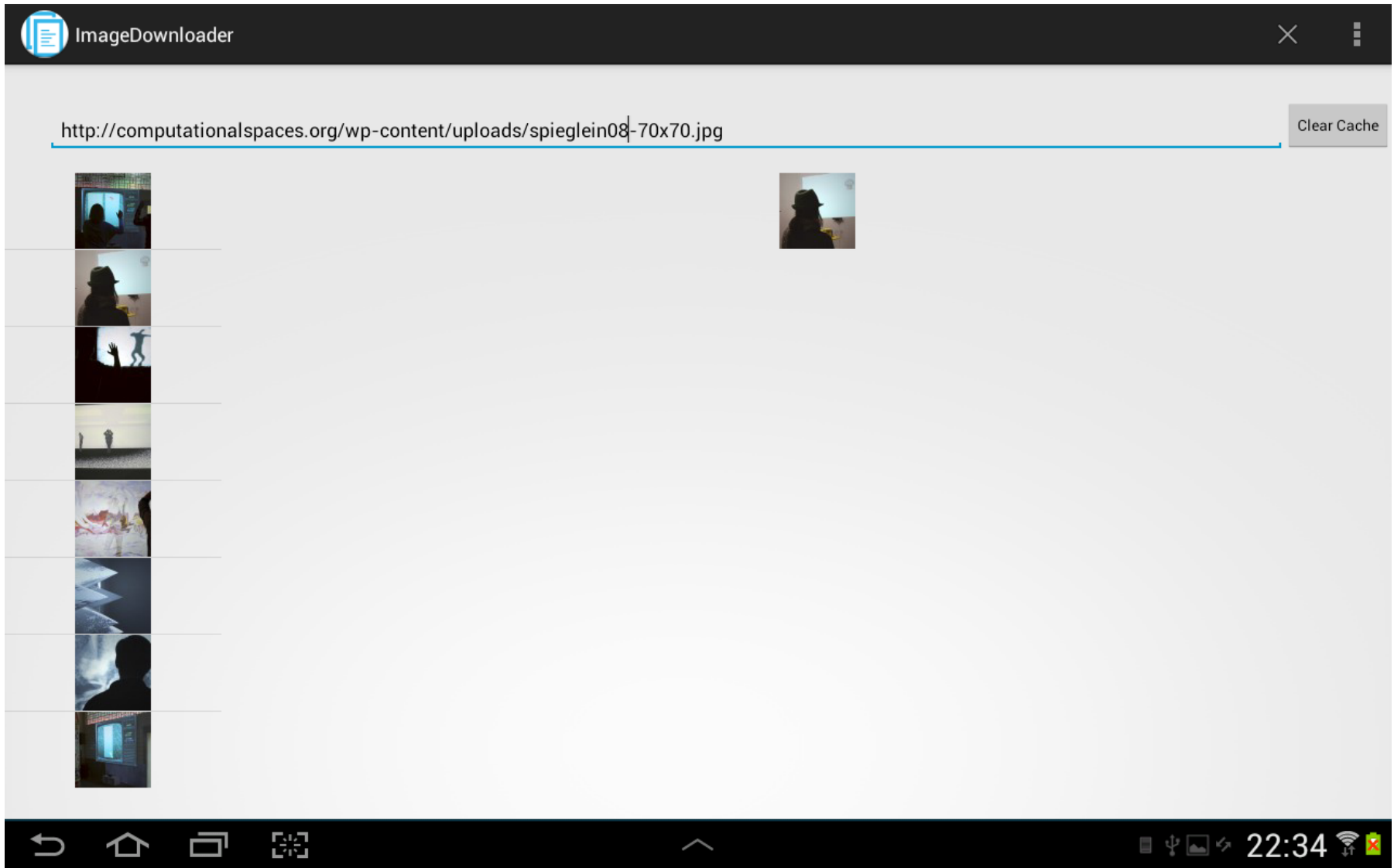
```
try {
    InputStream fis = getResources().
        openRawResource(
            R.raw.static_read_only);
    BufferedReader bfis = new
        BufferedReader(
            new InputStreamReader(fis));
    updateFieldsFrom(bfis);
    bfis.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

# Dateien- Applikations-Cache

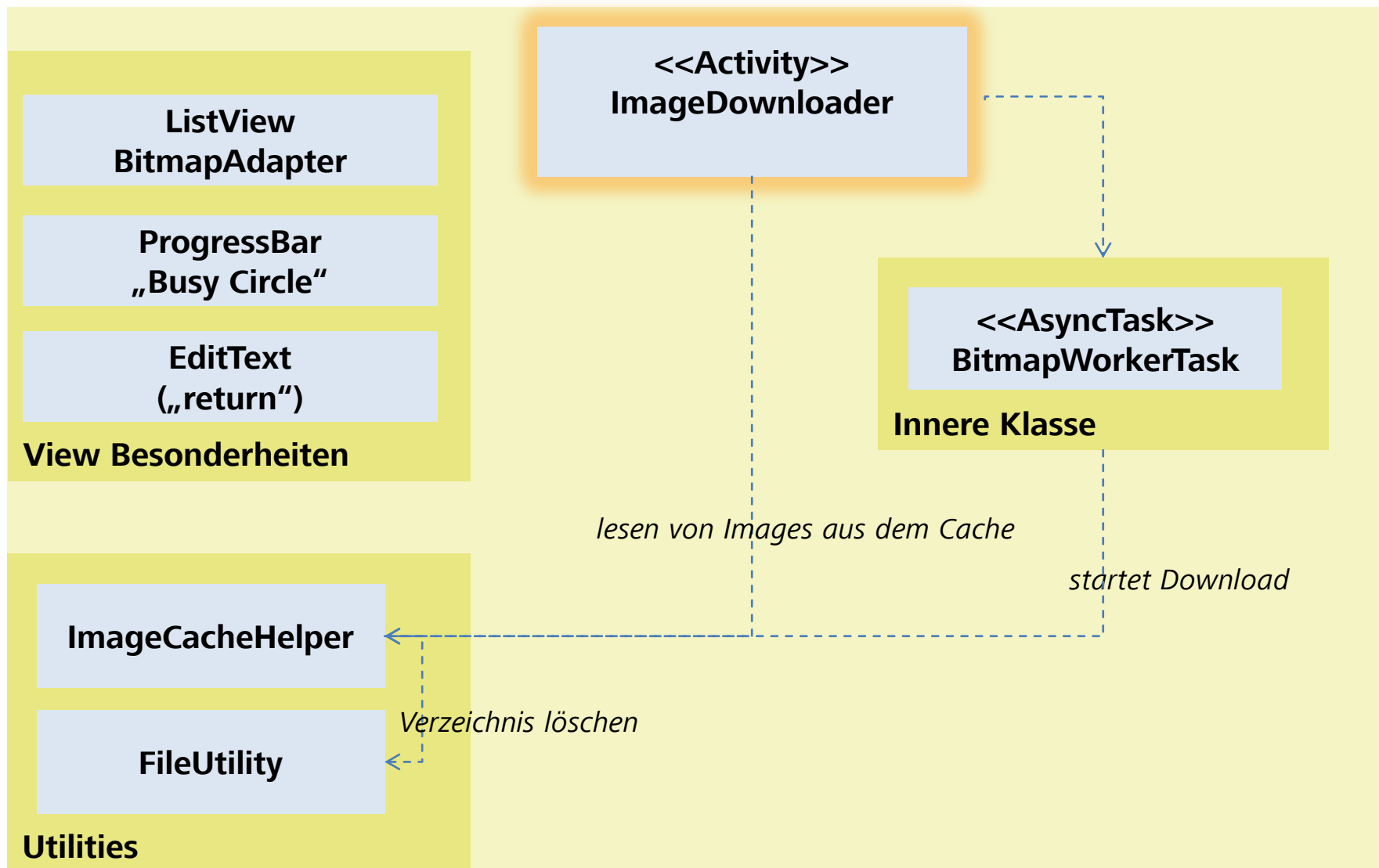
# Caching

- Ablegen von Daten, die zur Laufzeit einer Applikation benötigt werden,
- Zugriff auf das Cache Verzeichnis: *getCacheDir* → *File* (repräsentiert das Cacheverzeichnis der Applikation)
- Löschen des Cacheverzeichnisses durch Android
  - bei Speichermangel
  - bei der Deinstallation einer Applikation
- Explizites Löschen
  - über das Einstellungsmenü
  - programmatisch

# Cachen von Image-Downloads



# Überblick über die Lösung



# Downloads über BitmapWorkerTask

```
private class BitmapWorkerTask extends AsyncTask<String, Void, Bitmap> {
    private WeakReference<ListView> listViewReference;
```

```
    public BitmapWorkerTask(ListView listView) {
        listViewReference = new WeakReference<ListView>(listView);
    }
```

```
// params[0]: Url String of File params[1]: Cache Dir
```

```
protected Bitmap doInBackground(String... params) {
    busyWorker(View.VISIBLE);
    Bitmap bmp = ImageCacheHelper.downloadImageToCache(params[0], params[1]);
    busyWorker(View.INVISIBLE);
    return bmp;
}
```

Laden von Images  
in den Cache

```
protected void onPostExecute(Bitmap bitmap) {
    if (listViewReference != null && bitmap != null) {
        final ListView listView = listViewReference.get();
        if (listView != null) {
            BitmapAdapter adapter = (BitmapAdapter) listView
                .getAdapter();
            adapter.add(bitmap); } }
    }
}
```

Hinzufügen neuer Bitmaps  
über den Adapter

# Download Animation (1)

```
private class BitmapWorkerTask extends AsyncTask<String, Void, Bitmap> {

    // params[0]: Url String of File params[1]: Cache Dir
    protected Bitmap doInBackground(String... params) {
        busyWorker(View.VISIBLE);
        Bitmap bmp = ImageCacheHelper.downloadImageToCache(params[0], params[1]);
        busyWorker(View.INVISIBLE);
        return bmp;
    }

    private void busyWorker(final int visibility) {
        ImageDownloader.this.runOnUiThread(new Runnable() {

            public void run() {
                busyCircle.setVisibility(visibility);
            }
        });
    }
}
```

Delegation des  
Runnable an  
die Queue des  
UI Threads



## Download Animation (2)

<ProgressBar

```
    android:id="@+id/busyCircle"  
    style="?android:attr/progressBarStyle"  
    android:layout_width="40dp"  
    android:layout_height="40dp"  
    android:layout_alignParentLeft="true"  
    android:layout_marginTop="40dp"  
    android:indeterminate="true"  
    android:visibility="invisible" />
```

*indeterminate →  
Animation „busy circle“*

# Image (Bitmap) Download

```
public static Bitmap downloadImageToCache(String imageURL, String cacheDir) {
    Bitmap bitmap = null;
    try {
        URL url = new URL(imageURL);

        String filePart = url.getFile();
        String fileName = filePart.substring(filePart.lastIndexOf("/") + 1);

        File file = new File(getImageCacheDir(new File(cacheDir)), fileName);

        URLConnection ucon = url.openConnection();
        InputStream is = ucon.getInputStream();
        bitmap = BitmapFactory.decodeStream(is);

        FileOutputStream fos = new FileOutputStream(file);
        bitmap.compress(Bitmap.CompressFormat.PNG, 70, fos);
        fos.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
    return bitmap;
}
```

*liest Bitmap vom InputStream*

*schreibt Bitmap  
in eine Datei  
im Cache*

# Lesen von Bitmaps aus dem Cacheverzeichnis

```
public static Bitmap loadBitmapFromCache(File imageFile) throws FileNotFoundException
{
    try {
        FileInputStream fis = new FileInputStream(imageFile);
        return BitmapFactory.decodeStream(fis);

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    return null;
}
```

liest Bitmap  
von einer  
Datei

Fehler in der vollen  
Ausbaustufe nicht  
lokal behandeln →  
nach außen geben

# Lesen aller Bitmaps des Cache Verzeichnisses

```
public static Bitmap[] retrieveBitmapsFromCache(File cacheDir) {  
    File[] files = getImageCacheDir(cacheDir).listFiles();  
  
    Bitmap[] bitmaps = new Bitmap[files.length];  
    for (int i = 0; i < bitmaps.length; i++) {  
        try {  
            bitmaps[i] = LoadBitmapFromCache(files[i]);  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
    return bitmaps;  
}
```

Dateien des Cache  
verzeichnisses für  
Images lesen

# Löschen des Cacheverzeichnisses

```
public static boolean deleteCache(File cacheDir) {  
    File imageCacheDir = getImageCacheDir(cacheDir);  
    if (imageCacheDir.isDirectory()) {  
        File[] files = imageCacheDir.listFiles();  
        for (File file : files) {  
            boolean success = FileUtility.deleteDir(file);  
            if (!success)  
                return false;  
        }  
    }  
    return true;  
}
```

Dateien des Cache  
verzeichnisses löschen

```
public class FileUtility {  
    public static boolean deleteDir(File dir) {  
        if (dir.isDirectory()) {  
            File[] files = dir.listFiles();  
            for (File file : files) {  
                boolean success = deleteDir(file);  
                if (!success)  
                    return false;  
            }  
        }  
        return dir.delete();  
    }  
}
```

rekursives Löschen  
Verzeichnisse müssen  
leer sein

# View Besonderheiten

## EditText

```
urlField.setOnEditorActionListener(new TextView.OnEditorActionListener() {
```

Registrieren eines  
OnEditorActionListener

```
public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
```

```
    if (actionId == EditorInfo.IME_NULL && event.getAction() == KeyEvent.ACTION_DOWN) {
```

IME (InputMethod)  
Kombination entspricht  
„enter“ „return“

```
        String urlStr = urlField.getText().toString();
```

```
        if (!ImageCacheHelper.inCacheDir(getCacheDir(), urlStr)) {
```

```
            new BitmapWorkerTask(imageListView).execute(
                new String[] { urlStr, getCacheDir().toString() });
```

```
        } else {
```

```
            imageView.setImageBitmap(readFromCache(urlStr));
```

```
        }
```

```
    }
```

```
    return true;
```

```
}
```

```
});
```

# Android IMF

## Input Method Framework

- ➔ <http://grail.cba.csuohio.edu/~matos/notes/cis-493/lecture-notes/Android-Chapter07B-Hard-Soft-Keyboard-IMF.pdf>
- Android r1.5 introduced the notion of **Input Method Framework (IMF)**. *The idea is to let the IMF arbitrate the interaction between applications and the current input method chosen by the user.*

*The motivation behind this framework is the realization that as Android matures, more hardware /software devices, and input techniques will appear in user's applications, for instance:*

- real & virtual keyboards,
  - voice recognition,
  - hand writing,
  - etc...

# View Besonderheiten

## ListView Adapter für Bitmaps

```
public class BitmapAdapter extends ArrayAdapter<Bitmap> {

    public BitmapAdapter(Context context, int textViewResourceId, Bitmap[] bmps) {
        super(context, textViewResourceId, new ArrayList<Bitmap>(Arrays.asList(bmps)));
    }
```

*modifizierbare ArrayList erzeugen*

```
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;
        if (v == null) {
            LayoutInflater vi = (LayoutInflater) getContext().getSystemService(
                Context.LAYOUT_INFLATER_SERVICE);
            v = vi.inflate(R.layout.items_list_item, null);
        }
        Bitmap bmp = getItem(position);
        if (bmp != null) {
            ImageView iv = (ImageView) v.findViewById(R.id.list_item_image);
            if (iv != null) {
                iv.setImageBitmap(bmp);
            }
        }
        return v;
    }
```

*View für ein Item expandieren*

*eine ImageView pro Bitmap erzeugen  
Bitmap eintragen*

*Item View zurückgeben*



# ListView Adapter mit skalierten Bitmaps

```
public class BitmapAdapter extends ArrayAdapter<Bitmap> {  
    public View getView(int position, View convertView, ViewGroup parent) {  
        View v = convertView;  
        ...  
        Bitmap it = getItem(position);  
        if (it != null) {  
            ImageView iv = (ImageView) v.findViewById(R.id.list_item_image);  
            int[] scaleXY = calculateScaleXY(it, 200);  
  
            Bitmap bmp = Bitmap.createScaledBitmap(it, scaleXY[0], scaleXY[1], false);  
            if (iv != null) {  
                iv.setImageBitmap(bmp);  
            }  
        }  
        return v;  
    }  
}
```

Breite / Höhe für Zielbreite 200 umrechnen

Bitmap skalieren

# Effizientes Bearbeiten von Bitmaps

➔ <http://developer.android.com/training/displaying-bitmaps/index.html>

- Effizientes Laden großer Bitmaps über **BitmapFactory**
  - durch vorheriges Ermitteln der Größe
  - durch „Downsampling“ auf eine Zielgröße
- Bearbeiten von Bitmaps in eigenen Thread (➔ AsyncTasks)
- Caching
  - In-Memory: **LruCache**
  - Spezieller Platten LruCache: **DiskLruCache**

# Dateien

## Verwenden des externen Speichers

# Zugriff auf externen Speicher

- Berechtigungen:

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

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

# Prüfen auf Verfügbarkeit

```
/* 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;
}
```

# Public / Private Dateien

- Public:
  - bleiben erhalten, wenn die Applikation deinstalliert wird
- Private:
  - Beispiel: Downloads / tempäre Media Dateien
  - gehören zu einer Applikation
  - sollten gelöscht werden, wenn die Applikation deinstalliert wird (→ Android)
  - erreichbar für den Benutzer und andere Applikationen

# Public / Private Dateien

```
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;  
}
```

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