Block 30 – Dateioperationen

- Einführung
- Shared preferences
  - Schreiben
  - Lesen
- Interner Speicher
  - Schreiben
  - Lesen
- Externer Speicher
- Installationsunabhängige Daten
- Verzeichnisse

Block 30 – Lernziele

In diesem Block werden Sie lernen, ...

- wie Sie einzelne Werte über das Anwendungsende hinaus in Shared Preferences speichern können.
- wie Sie einzelne Werte nach dem Anwendungsstart wiederherstellen können.
- wie sich Daten in Dateien in den internen und externen Speicher schreiben und lesen lassen.
- wie sich der interne und externe Speicher unterscheidet.
- wie Dateien auch nach der Deinstallation der mobilen Anwendung verfügbar bleiben.
- wie zwischen Ordnern und Verzeichnissen navigiert werden kann.

30.1 Einführung

Im Block 'Multimedia' wurden bereits einige Programmabschnitte gezeigt, mit welchen Dateien auf der Android-Plattform angelegt wurden.

Der Android Software Stack (Block 'Einführung') zeigt ebenfalls an, dass die Android-Plattform als vollwertiges Betriebssystem über ein ausgereiftes Dateiverwaltungssystem (File System) verfügt.

Auch wenn die Ressourcen einer Anwendung über den Build-Mechanismus verfügbar sind, und hoch dynamische Daten in Datenbanken liegen, so gibt es zahlreiche Anwendungsfälle, für die eine Ablage von Daten in Dateien notwendig ist.

30.2 Shared Preferences

Android bietet über *SharedPreferences* einen Mechanismus an, mit welchem Einstellungen dauerhaft abgespeichert werden können.

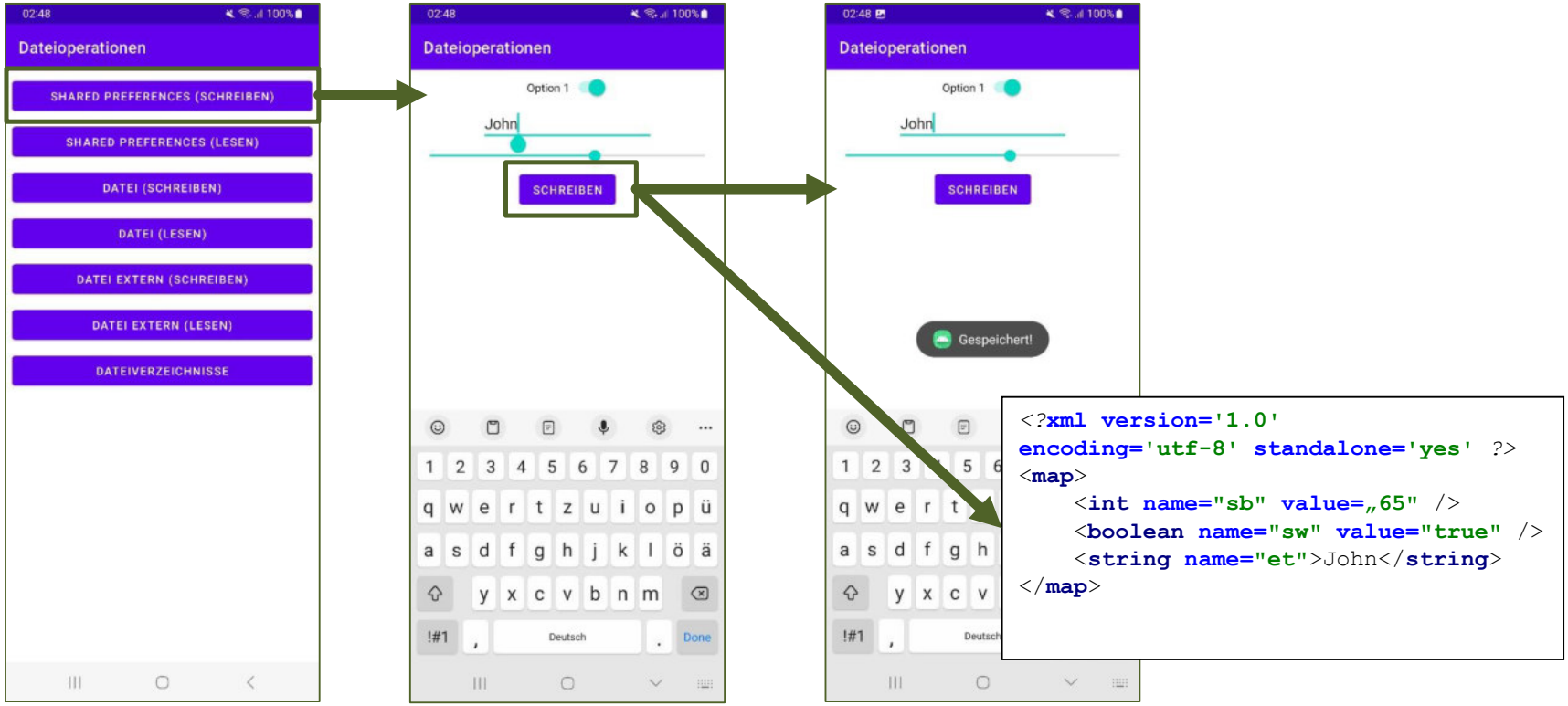
Es lassen sich alle primitiven Datentypen (`boolean`, `float`, `int`, `long`, `String`) in den Shared Preferences speichern.

Die Speicherung wird über den Anwendungszyklus hinaus und auch über den Neustart des mobilen Geräte hinaus gewährleistet.

Die Shared Preferences sind auch vor dem Zugriff anderer Anwendungen geschützt.

Intern löst Android eine Persistierung der Key/Value-Paare durch eine XML-Datei aus. Diese kann ggf. auch zur Übertragung von Konfigurationseinstellungen für andere Installationen derselben mobilen Anwendung genutzt werden.

30.2.1 Shared Preferences - Schreiben



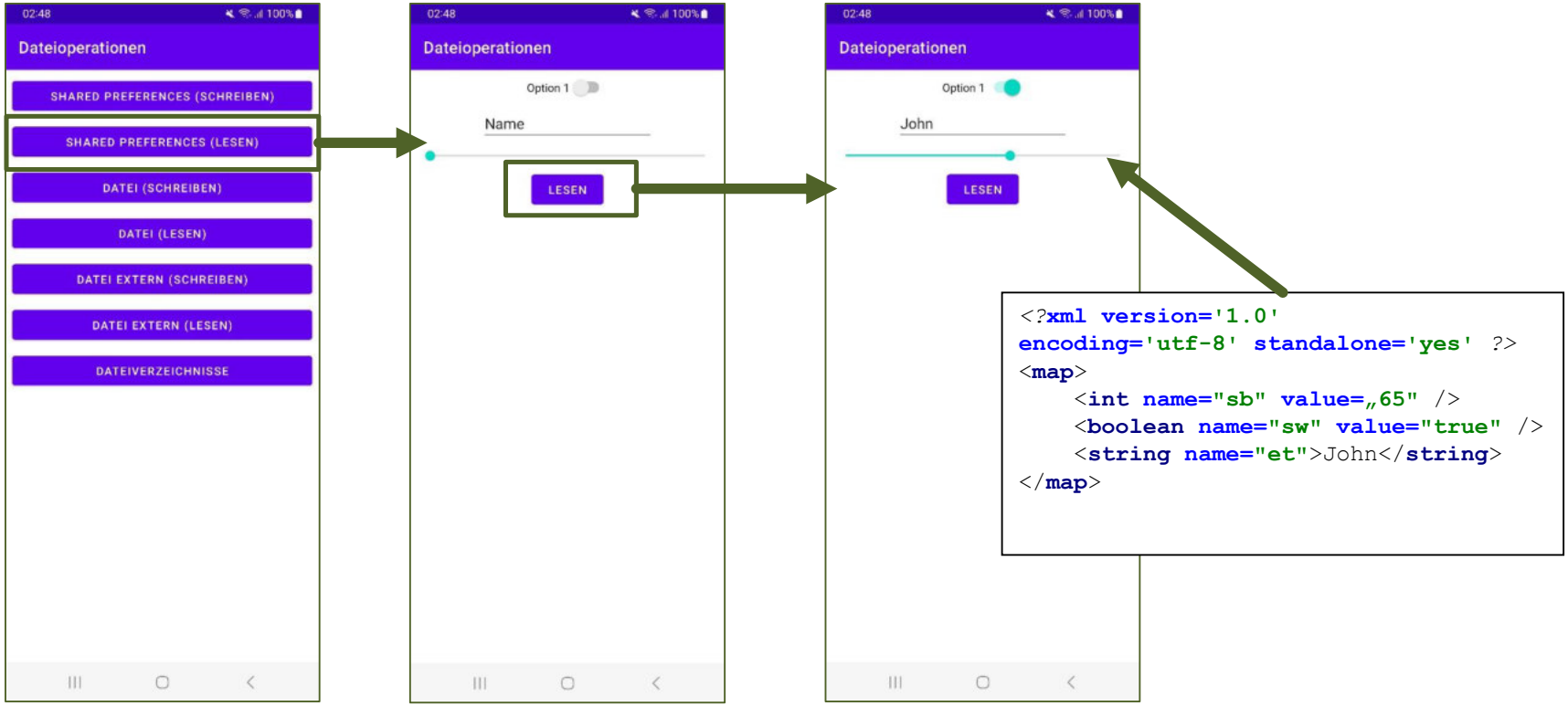
30.2.1 Shared Preferences - Schreiben (Forts.)

Über die Methode `getSharedPreferences()` können die Shared Prefences für die aktuelle mobile Anwendung bezogen werden. Für den schreibenden Zugriff ist ein `SharedPreferences.Editor` notwendig.

```
val ocl: View.OnClickListener = object : View.OnClickListener {
  override fun onClick(view: View?) {
    val sp = getSharedPreferences(PREFS_NAME, 0)
    val sp_editor = sp.edit()
    sp_editor.putBoolean("sw", sw.isChecked())
    sp_editor.putString("et", et.getText().toString())
    sp_editor.putInt("sb", sb.getProgress())
    sp_editor.commit()
    val toast = Toast.makeText(applicationContext,
      "Gespeichert!", Toast.LENGTH_SHORT)
    toast.show()
  }
}
```

Abschließend müssen alle Änderungen mit der Methode `commit()` bestätigt werden.

30.2.2 Shared Preferences - Lesen



30.2.2 Shared Preferences - Lesen (Forts.)

Mit typspezifischen Getter-Methoden können die zuvor gespeicherten Shared Preferences wieder abgerufen werden. Dabei steht der erste Parameter (String) für den Key des Wertes und der zweite für einen Standard-Rückgabewert. Der Standard-Rückgabewert wird immer dann zurückgegeben, wenn der Schlüssel nicht gefunden werden konnte.

```
val ocl: View.OnClickListener = object : View.OnClickListener {
  override fun onClick(view: View?) {
    val sp = getSharedPreferences(PREFS_NAME, 0)
    sw.setChecked(sp.getBoolean("sw", false))
    et.setText(sp.getString("et", "Kein Wert gefunden!"))
    sb.setProgress(sp.getInt("sb", 0))
  }
}
```

30.3 Interner Speicher

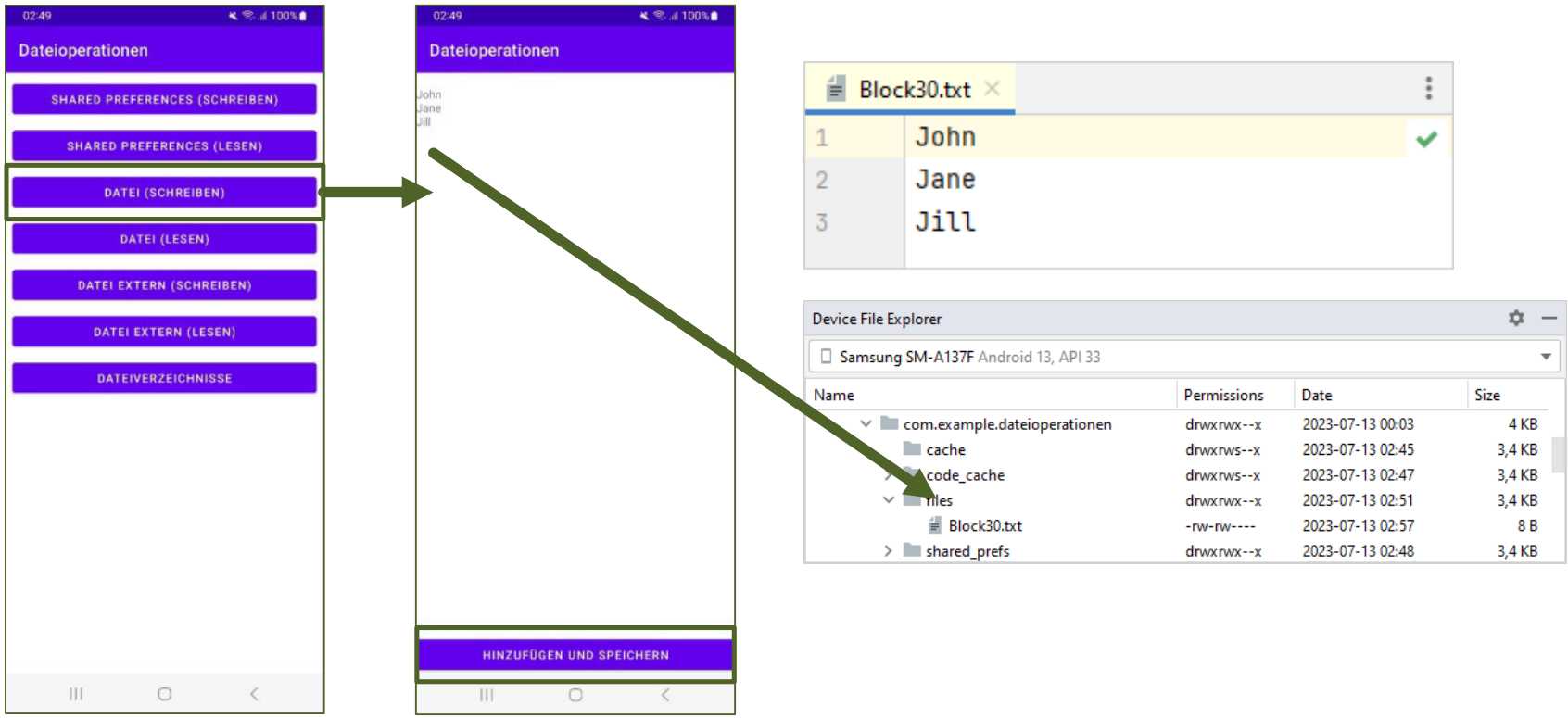
Da die Shared Preferences als Key/Value-Paare unpraktisch für große Datenmengen sind, können Dateien zusätzlich in beliebigen Formaten erzeugt werden.

Alle hier gezeigten Dateioperationen nutzen den Standardspeicherort für die jeweilige mobile Anwendung.

Die Dateien am Standardspeicherort überdauern zwar den Anwendungszyklus bis zum Neutstart der Anwendung als auch den Neutstart des Gerätes, werden aber bei einer Deinstallation der mobilen Anwendung ebenfalls gelöscht werden.

Der Zugriff auf andere Speicherorte setzt eine entsprechende Berechtigung voraus (nicht empfohlen) bzw. ist ohne spezielle Rechte (Root-Rechte) gar nicht möglich.

30.3.1 Interner Speicher - Schreiben



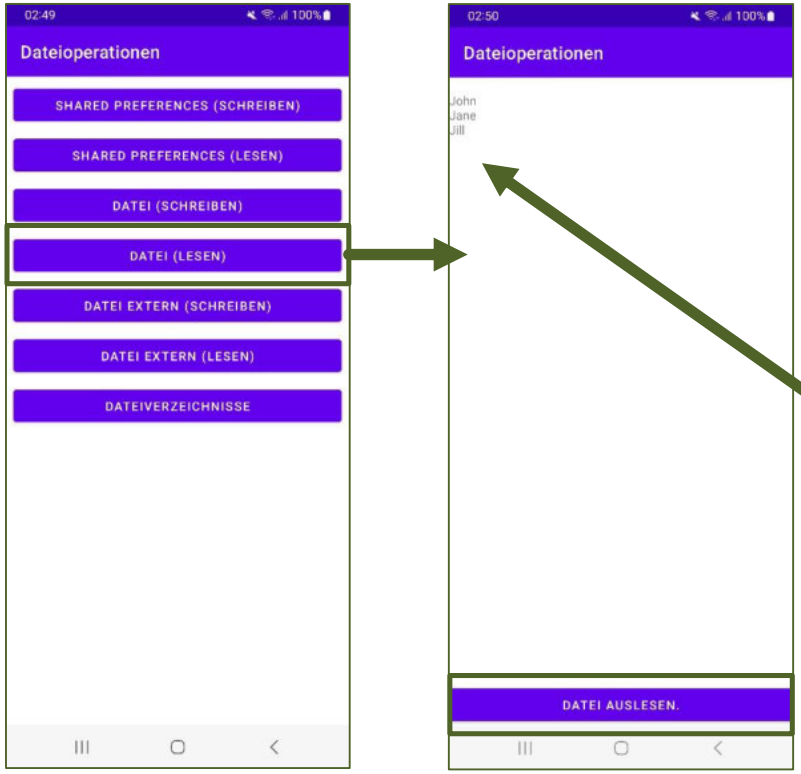
30.3.1 Interner Speicher - Schreiben (Forts.)

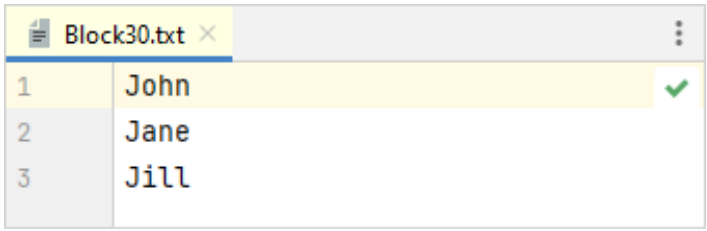
Die Methode `openFileOutput()` öffnet eine neue Datei zur Ausgabe, welche in einem anwendungsspezifischen Bereich liegt. Das erste Parameter gibt dabei den Dateinamen an. Der zweite Parameter regelt die Sichtbarkeit hinsichtlich des Zugriffs weiterer mobiler Anwendungen.

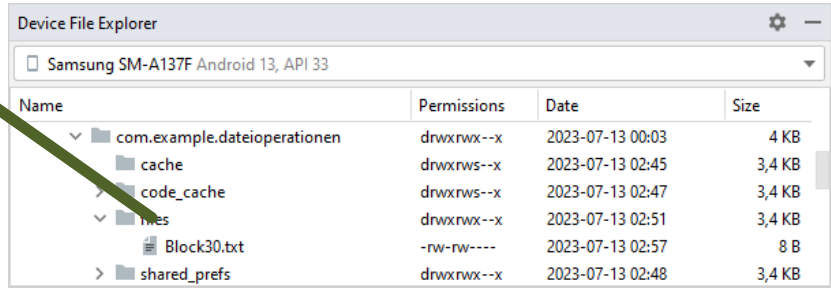
```
val FILENAME = "Block30.txt"

val ocl: View.OnClickListener = object : View.OnClickListener {
  override fun onClick(view: View?) {
    ...
    try {
      val fos: FileOutputStream = openFileOutput(FILENAME,
        MODE_PRIVATE)
      fos.write(tv.getText().toString().toByteArray())
      fos.close()
    } catch (e: IOException) {
      e.printStackTrace()
    }
  }
}
```

Die Daten werden dann Byte-weise in die Datei übertragen und der Stream wird geschlossen.

30.3.2 Interner Speicher - Lesen







30.3.2 Interner Speicher - Lesen (Forts.)

Die Methode `openFileInput()` öffnet die Datei zum Einlesen, welche in einem anwendungsspezifischen Bereich liegt.

```
val FILENAME = "Block30.txt"

val ocl: OnClickListener = object : OnClickListener {
  override fun onClick(view: View?) {
    try {
      val fis: FileInputStream = openFileInput(FILENAME)
      val length: Int = fis.available()
      var result = ""
      for (i in 0 until length) {
        result = result + fis.read().toChar()
      }
      tv.setText(result)
      fis.close()
    } catch (e: IOException) {
      e.printStackTrace()
    }
  }
}
```

Hinweis: Für den lesenden und schreibenden Zugriff auf den privaten Bereich sind keine Berechtigungen erforderlich.

30.4 Externer Speicher

Jedes auf Android basierende System (z.B. auch TV) unterstützt einen externen Speicher auf welchem ebenfalls Dateien geschrieben bzw. gelesen werden können.

Der externe Speicher muss dabei nicht entfernbar sein (z.B. USB-Stick, SD-Karte), sondern kann auch fest verbaut sein.

Der externe Speicher ist systemweit verfügbar bzw. kann auch für einen Host über USB verfügbar gemacht werden (z.B. Zugriff auf Fotos, Music, etc.).

Im Unterschied zum internen Speicher gelten für den externen Speicher gewisse Zustände: Mounted, Missing, Read-only, etc.

30.4 Externer Speicher (Forts.)

Vor dem Zugriff auf den externen Speicher sollte der Zustand des Speichers überprüft werden:

```
var mExternalStorageAvailable = false
var mExternalStorageWriteable = false
var state: String = Environment.getExternalStorageState()

private fun checkState() {
  if (Environment.MEDIA_MOUNTED.equals(state)) {
    mExternalStorageWriteable = true
    mExternalStorageAvailable = mExternalStorageWriteable
  } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    mExternalStorageAvailable = true
    mExternalStorageWriteable = false
  } else {
    mExternalStorageWriteable = false
    mExternalStorageAvailable = mExternalStorageWriteable
  }
}
```

30.4 Externer Speicher (Forts.)

Der Zugriff auf den externen Speicher erfolgt analog zu dem Zugriff auf den internen Speicher, jedoch ist eine Anpassung vorzunehmen:

```
val sdcard: File? = getExternalFilesDir(null)
val file = File(sdcard, FILENAME)
val fis = FileInputStream(file)
...
```

Der Dateipfad wird nunmehr über die Methode `getExternalFilesDir()` gebildet.

Hinweis: Auch hier sind für den privaten Bereich keine Berechtigungen notwendig.

30.4 Externer Speicher (Forts.)

Der Aufruf der Methode `getExternalFilesDir()` mit dem Paramter `null` führt zu dem anwendungsspezifischen Speicherort der mobilen Anwendung, welcher für beliebige Daten geeignet ist.

Es können mit diesem Parameter jedoch auch Ordner für spezifische Daten erzeugt werden:

DIRECTORY_MUSIC für Musikdateien
DIRECTOR_RINGTONES für Klingeltöne
usw.

Der unter Android laufende Media Scanner erkennt diese Ordner (welche spezielle Namen tragen) und fügt die darin enthaltenen Daten den entsprechenden Anwendungen zu (z.B. Musikdateien dem Music Player, usw.)

Hinweis: Unabhängig von dem o.g. Parameter werden alle Dateien bei der Deinstallation der mobilen Anwendung ebenfalls gelöscht.

30.5 Installationsunabhängige Daten

Die Methode `Environment.getExternalStoragePublicDirectory()` liefert die öffentlichen Speicherorte zurück.

Mit den bereits bekannten Argumenten können die unterschiedlichen Ordnertypen angefragt werden:

DIRECTORY_MUSIC für Musikdateien
DIRECTOR_RINGTONES für Klingeltöne
usw.

Eine Speicherung an diesen Speicherorten bleibt von einer möglichen Deinstallation der mobilen Anwendung erhalten.

30.6 Verzeichnisse

Zur Bearbeitung von Verzeichnissen stehen ebenfalls Methoden bereit:

`getFilesDir()`:Liefert den absoluten Pfad eines internen Speicherorts.

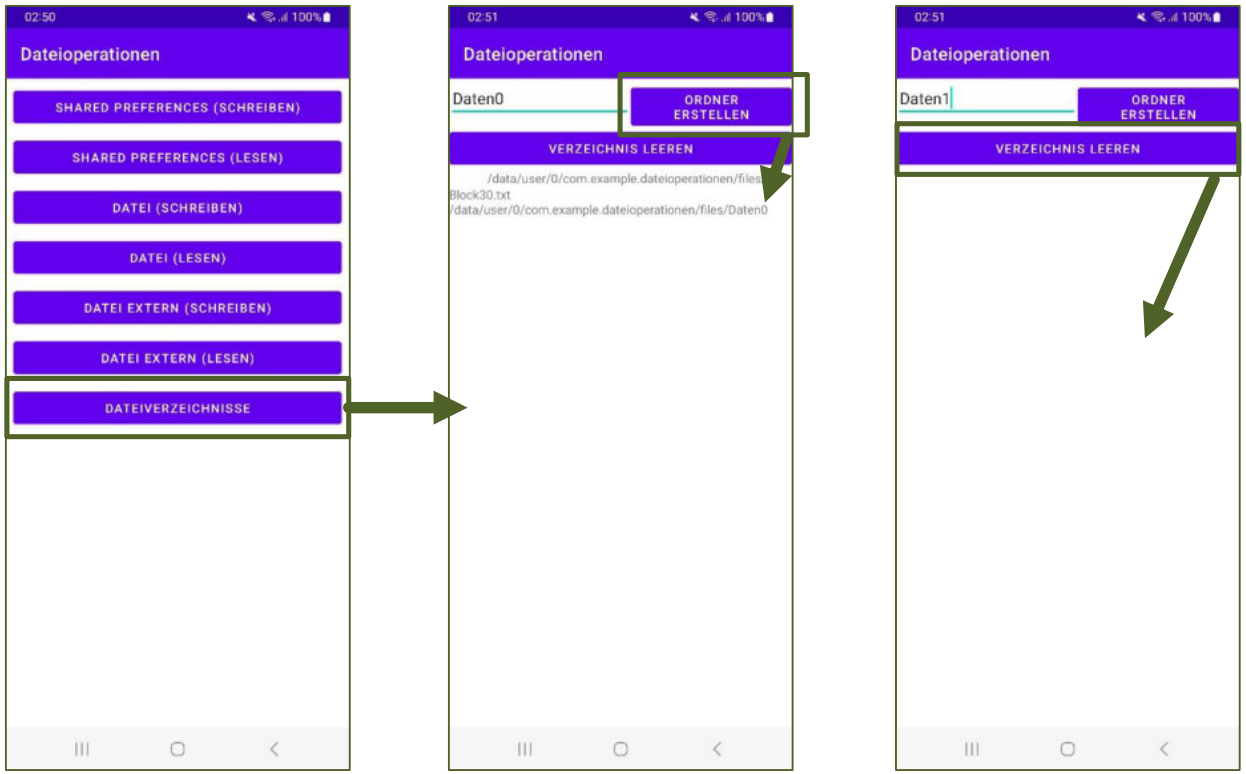
`getDir()`:Erzeugt (oder öffnet einen existierenden) Speicherort innerhalb des internen Speichers.

`deleteFile()`:Löscht eine Datei innerhalb des internen Speicherorts.

`fileList()`:Liefert eine Liste der Dateien (und Ordner) innerhalb des internen Speicherorts

Hinweis: In den gezeigten Beispielen wurde die Datei bei dem Schreibvorgang immer implizit erzeugt.

30.6 Verzeichnisse (Forts.)



Block 30 – Zusammenfassung

- Geringe Datenmengen und Einstellung einer mobilen Anwendung können über **Shared Preferences** gespeichert werden
- Der **Interne Speicher** erlaubt das lokale Speichern von Daten über den Anwendungszyklus hinaus
- Der **Externe Speicher** erlaubt die Speicherung analog zum internen Speicher, jedoch sind die **Zustände** des Speichermedium zu beachten
- Ohne besondere Vorkehrung werden gespeicherte Dateien zusammen mit der Anwendung deinstalliert d.h. **gelöscht**

Block 30 – Weitere Aufgaben

- Erstellen Sie einen kleinen Datei-Explorer, der vorhandenen Dateien in internem und externenen Ordnern anzeigen kann.
- Erweitern Sie den Datei-Explorer, so dass Dateien kopiert und ggf. Ordner erstellt werden können.
- Der Datei-Explorer sollte auch Metadaten wie z. B. die Dateigrößen oder das Speicherdatum anzeigen können.

Block 30 – Literatur I