Programowanie urządzeń mobilnych

dr inż. Andrzej Grosser
na podstawie wykładu
dr inż. Juliusza Mikody

SharedPreferences

- Umożliwia przechowywanie małej ilości danych typów podstawowych
- Dane są przechowywane w postaci dostępnego pomiędzy sesjami aplikacji słownika.
- Używane przede wszystkim do zapamiętywania dostosowanych do preferencji użytkownika danych.

SharedPreferences

- Do uzyskiwana obiektu klasy SharedPreferences służą dwie metody:
  - Activity.getPreferences(int mode)
  - Context.getSharedPreferences(String nazwa, int mode) – nazwa jest nazwą pliku
- Wartości z obiektu uzyskuje się za pomocą metod getXX() np.: getString(String klucz, String wartDomyslna), getInt(String klucz, int wartDomyslna).

SharedPreferences

- Zapisywanie danych jest wykonywane z wykorzystaniem metody SharedPreferences.edit() - zwraca ona obiekt klasy SharedPreferences.Editor
- Wartości są zapisywane z wykorzystaniem metod klasy edytora np.:
  - putInt(String klucz, int wartość)
  - remove(String klucz)
- Modyfikacje są zatwierdzane za pomocą metody commit()

Baza danych SQLite

- SQLite – to system zarządzania bazą danych obsługujący język SQL (ang. Structured Query Language)
- Biblioteka implementuje silnik SQL, dając możliwość używania bazy danych bez konieczności uruchamiania osobnego procesu RDBMS.
- Bazy danych zapisywane są jako pliki binarne (jeden plik do 2TB).
- SQLite obsługuje między innymi: zapytania zagnieżdżone, widoki, klucze obce, transakcje, wyzwalacze (częściowo).

Baza danych SQLite

- Baza udostępnia kilka typów podstawowych:
  - INTEGER (1 do 8 bajtów) INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8,
  - TEXT – typ tekstowy - VARCHAR(255), CLOB,
  - NONE – typ nieokreślony BLOB,
  - REAL – typ zmienno-przecinkowy – REAL, DOUBLE, DOUBLE PRECISION, FLOAT,
  - NUMERIC – typ stałoprzecinkowy – NUMERIC, DECIMAL(10,5), BOOLEAN, DATE, DATETIME.

Baza danych SQLite

- `CREATE TABLE t1( t TEXT, nu NUMERIC, i INTEGER, r REAL, no BLOB );`

- `INSERT INTO t1 VALUES('500.0', '500.0', '500.0', '500.0', '500.0');`
- `SELECT typeof(t), typeof(nu), typeof(i), typeof(r), typeof(no) FROM t1;`
- `text|integer|integer|real|text`

- `INSERT INTO t1 VALUES(500, 500, 500, 500, 500);`
- `SELECT typeof(t), typeof(nu), typeof(i), typeof(r), typeof(no) FROM t1;`
- `text|integer|integer|real|integer`

Tworzenie bazy danych

- Bazy danych projektu zapisywane są w katalogu: **`/data/data/*/databases/name.db`**
- Znak * zastępuje nazwa pakietu programu tworzącego bazę danych,
- Usuniecie pliku db powoduje usuniecie całej bazy danych,
- Bazę danych można utworzyć:
  - bezpośrednio: Context.openOrCreateDatabase
  - z użyciem klasy: SQLiteOpenHelper

Tworzenie bazy danych

```
public class Pojazdy extends SQLiteOpenHelper {
  private static final int version = 1;
  public Pojazdy(Context context) {
    super(context, "pojazdy.db", null, version);
  }

  public void onCreate(SQLiteDatabase baza) {
    try{
      baza.execSQL("CREATE TABLE IF NOT EXISTS Pojazdy "
        + "(marka VARCHAR, model VARCHAR)");
      Log.v("Pojazdy", "Baza została utworzona");
    } catch(SQLiteException e) {
      Log.e("Pojazdy", "Błąd tworzenia lub otwarcia bazy danych");
    }
  }

  public void onUpgrade(SQLiteDatabase baza, int ver1, int ver2) {
    Log.v("Pojazdy", "Zmiana wersji bazy " + ver1 + " " + ver2);
    baza.execSQL("drop table Pojazdy");
    onCreate(baza);
  }
}
```

Zarządzanie bazą danych

- Tworzenie obiektu bazy danych:

```
Pojazdy p = new Pojazdy(this);
```

- Otwarcie bazy danych do odczytu:

```
SQLiteDatabase db =
    p.getReadableDatabase();
```

- Otwarcie bazy danych do zapisu:

```
SQLiteDatabase db =
    p.getWritableDatabase();
```

Zarządzanie bazą danych

```
Pojazdy p = new Pojazdy(this);
SQLiteDatabase db = p.getWritableDatabase();

// wstawianie z uwzględnieniem kolumn
db.execSQL("INSERT INTO Pojazdy (marka, model) "
                        "VALUES('Toyota','Corolla');");

// kolejność pól bazy danych
db.execSQL("INSERT INTO Pojazdy VALUES('Fiat','Brava');");

// z użyciem parametrów
db.execSQL("INSERT INTO Pojazdy VALUES(?,?);",
          new String[]{"Karlik","20"});

// Z użyciem zbioru wartości
ContentValues cv = new ContentValues();
cv.put("marka", "Citroen");
cv.put("model", "C5");
db.insert("Pojazdy", null, cv);
```

Zarządzanie bazą danych

- Odczyt danych z tabeli:

```
Pojazdy p = new Pojazdy(this);
SQLiteDatabase db = p.getReadableDatabase();
Cursor cursor = db.rawQuery(
          "SELECT marka, model FROM Pojazdy", null);
if(cursor.moveToFirst()){
    do {
        String marka = cursor.getString(
            cursor.getColumnIndex("marka"));
        String model = cursor.getString(
            cursor.getColumnIndex("model"));
        Log.i(marka, model);
    } while(cursor.moveToNext());
}

cursor.close();
db.close();
p.close();
```

Zarządzanie bazą danych

- Odczyt wartości z bazy danych:
  - rawQuery (String sql, String[] selectionArgs)

sql – zapytanie, **selectionArgs** – dodatkowe argumenty (dla parametrów ?)

  - query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

Table – nazwa tabeli, **columns** – wybierane kolumny, **selection** – wybór wartości, **selectionArgs** - dodatkowe argumenty, **groupBy** – elementy grupujące, **having** – selekcja wartości grupowanych, **orderBy** – kolejność wyników, **limit** – górny zakres liczby wierszy wyniku.

Konsola – dostęp zewnętrzny

- android list avd - lista dostępnych urządzeń
- adb devices – uruchomione urządzenia
- adb shell – dostęp do konsoli urządzenia
- ls – R /data/data/*/databases
- sqlite3 /data/data/pl.pojazdy/databases/pojazdy.db
  - .tables
  - .schema
  - .quit

Dostawcy treści

- Opakowują dane, które są dostarczane do aplikacji z wykorzystaniem interfejsu ContentResolver.
- Wymuszają odpowiednie uprawnienia do dostępu do danych.
- Dostawcy treści są wymagani w sytuacji, gdy dane mają być współdzielone pomiędzy różnymi aplikacjami.

ContentResolver

- Udostępnia interfejs podobny do bazodanowego – np. query(), insert(), update() itp.
- Pozwala na korzystanie z dodatkowych serwisów, które pozwalają na powiadomienie zarejestrowanych obserwatorów o zmianie danych dostawcy treści.
- Podział na ContentResolver i ContenProvider pozwala na dostęp do danych w jednym procesie w innym procesie.

Standardowi dostawcy treści

- Browser – dane przeglądarki internetowej, np. zakładki
- CallLog – historia przeprowadzonych rozmów telefonicznych
- Contact (ContactsContract) – dane kontaktowe (numery telefoniczne itd.)
- MediaStore – dane multimedialne – zdjęcia, filmy itd.

Budowa własnego dostawcy treści

- W celu utworzenia dostawcy treści należy rozszerzyć klasę android.content.ContentProvider oraz zaimplementować metody:
  - query – pobieranie danych
  - insert – wprowadzanie informacji
  - update – zmiana danych
  - delete – usuwanie pozycji
  - getType – zwraca MIME dla danego URI

Następny przykład pochodzi z książki:
Android 2 – Tworzenie aplikacji
Sayed Hashimi, Satya Komatineni, Dave MacLean

Przykład – dostawca treści

```
public class BookProviderMetaData
{
    public static final String AUTHORITY = "pl.books.BookProvider";

    public static final String DATABASE_NAME = "book.db";
    public static final int DATABASE_VERSION = 1;
    public static final String BOOKS_TABLE_NAME = "books";

    private BookProviderMetaData() {}

    // wewnętrzna klasa opisująca obiekt BookTable
    // dzięki BaseColumns uzyskana jest kolumna _ID
    public static final class BookTableMetaData implements BaseColumns
    {
        private BookTableMetaData() {}
        public static final String TABLE_NAME = "books";

        // definicje identyfikatora URI oraz typu MIME
        public static final Uri CONTENT_URI =
            Uri.parse("content://" + AUTHORITY + "/books");
```

Przykład – dostawca treści

```
    public static final String CONTENT_TYPE =
        "vnd.android.cursor.dir/vnd.books";

    public static final String CONTENT_ITEM_TYPE =
        "vnd.android.cursor.item/vnd.books";

    public static final String DEFAULT_SORT_ORDER = "name DESC";

    // Tu rozpoczynają się dodatkowe kolumny.
    // typ string
    public static final String BOOK_NAME = "name";

    // typ string
    public static final String BOOK_ISBN = "isbn";

    // typ string
    public static final String BOOK_AUTHOR = "author";

  }
}
```

Przykład – dostawca treści

```
public class BookProvider extends ContentProvider
{
    // Tworzy mapę projekcji kolumn.
    private static HashMap<String, String> sBooksProjectionMap;
    static
    {
        sBooksProjectionMap = new HashMap<String, String>();
        sBooksProjectionMap.put(BookTableMetaData._ID,
                BookTableMetaData._ID);

        // kolumny name, isbn, author
        sBooksProjectionMap.put(BookTableMetaData.BOOK_NAME,
                BookTableMetaData.BOOK_NAME);
        sBooksProjectionMap.put(BookTableMetaData.BOOK_ISBN,
                BookTableMetaData.BOOK_ISBN);
        sBooksProjectionMap.put(BookTableMetaData.BOOK_AUTHOR,
                BookTableMetaData.BOOK_AUTHOR);
}
```

Przykład – dostawca treści

```
// Mechanizm umożliwiający identyfikowanie wzorców wszystkich
// przychodzących identyfikatorów URI.

    private static final UriMatcher sUriMatcher;
    private static final int
      INCOMING_BOOK_COLLECTION_URI_INDICATOR = 1;
    private static final int
      INCOMING_SINGLE_BOOK_URI_INDICATOR = 2;
static {
    sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    sUriMatcher.addURI(BookProviderMetaData.AUTHORITY,
        "books",
        INCOMING_BOOK_COLLECTION_URI_INDICATOR);

    sUriMatcher.addURI(BookProviderMetaData.AUTHORITY,
        "books/#",
        INCOMING_SINGLE_BOOK_URI_INDICATOR);
}
```

Przykład – dostawca treści

```
// Zajmuje się kwestią wywoływania zwrotnego metody OnCreate

    private DatabaseHelper mOpenHelper;

    public boolean onCreate() {
        mOpenHelper = new DatabaseHelper(getContext());
        return true;
    }

    private static class DatabaseHelper
        extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, BookProviderMetaData.DATABASE_NAME,
                null, BookProviderMetaData.DATABASE_VERSION);
        }
```

Przykład – dostawca treści

```
// Tworzy bazę danych
 public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE "
        + BookTableMetaData.TABLE_NAME + " ("
        + BookProviderMetaData.BookTableMetaData._ID
        + " INTEGER PRIMARY KEY,"
        + BookTableMetaData.BOOK_NAME + " TEXT,"
        + BookTableMetaData.BOOK_ISBN + " TEXT,"
        + BookTableMetaData.BOOK_AUTHOR + " TEXT"
        + ");");
    }

    // Zajmuje się kwestią zmiany wersji
    public void onUpgrade(SQLiteDatabase db,
        int oldVersion, int newVersion) {
        Log.w("BookProvider", "Aktualizacja " + oldVersion
            + " do wersji " + newVersion);
        db.execSQL("DROP TABLE IF EXISTS "
            + BookTableMetaData.TABLE_NAME);
        onCreate(db);
    }
}
```

Przykład – dostawca treści

```
@Override
public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
    case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
        return BookTableMetaData.CONTENT_TYPE;

    case INCOMING_SINGLE_BOOK_URI_INDICATOR:
        return BookTableMetaData.CONTENT_ITEM_TYPE;

    default:
        throw new IllegalArgumentException(
            "Nieznany ident. URI " + uri);
    }
}
```

Przykład – dostawca treści

```
public Cursor query(Uri uri, String[] projection,
    String selection, String[] selectionArgs,
    String sortOrder)
{
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(BookTableMetaData.TABLE_NAME);
    qb.setProjectionMap(sBooksProjectionMap);
    switch (sUriMatcher.match(uri)) {
        case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
            break;
        case INCOMING_SINGLE_BOOK_URI_INDICATOR:
            qb.appendWhere(BookTableMetaData._ID
                    + "=" + uri.getPathSegments().get(1));
            break;

        default:
            throw new IllegalArgumentException(
                "Nieznany ident. URI" + uri);
}
```

Przykład – dostawca treści

```
// Jeżeli kolejność sortowania nie jest określona,
// należy skorzystać z domyślnej wartości
String orderBy;
if (TextUtils.isEmpty(sortOrder)) {
    orderBy = BookTableMetaData.DEFAULT_SORT_ORDER;
} else {
    orderBy = sortOrder;
}
```

```
    // Otwiera bazę danych i uruchamia kwerendę
    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
    Cursor c = qb.query(db, projection, selection,
         selectionArgs, null, null, orderBy);

    // Mówi kursorowi, który identyfikator URI ma być
    // obserwowany na wypadek zmiany źródła danych.
    c.setNotificationUri(
         getContext().getContentResolver(), uri);
    return c;
}
```

Przykład – dostawca treści

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    // Sprawdza żądany identyfikator Uri
    if (sUriMatcher.match(uri) !=
         INCOMING_BOOK_COLLECTION_URI_INDICATOR) {
         throw new IllegalArgumentException(
             "Nieznany ident. URI " + uri);
    }
    if (values.containsKey(BookTableMetaData.BOOK_NAME) == false) {
         throw new SQLException(" Nieudana próba wstawienia wiersza "
             + "z powodu braku nazwy książki " + uri);
    }

    if (values.containsKey(BookTableMetaData.BOOK_ISBN) == false) {
         values.put(BookTableMetaData.BOOK_ISBN,
             "Nieznany numer ISBN");
    }
    if (values.containsKey(BookTableMetaData.BOOK_AUTHOR) == false) {
         values.put(BookTableMetaData.BOOK_ISBN, "Nieznany autor");
    }
```

Przykład – dostawca treści

```
SQLiteDatabase db = mOpenHelper.getWritableDatabase();

long rowId = db.insert(BookTableMetaData.TABLE_NAME,
    BookTableMetaData.BOOK_NAME, values);

if (rowId > 0) {
    Uri insertedBookUri = ContentUris.withAppendedId(
        BookTableMetaData.CONTENT_URI, rowId);
    getContext().getContentResolver().
        notifyChange(insertedBookUri, null);
    return insertedBookUri;
}

throw new SQLException(
    "Nieudana próba umieszczenia wiersza w " + uri);
}
```

Przykład – dostawca treści

```
public int update(Uri uri, ContentValues values, String where,
          String[] whereArgs){
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri)) {
        case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
            count = db.update(BookTableMetaData.TABLE_NAME,
                values, where, whereArgs);
            break;
        case INCOMING_SINGLE_BOOK_URI_INDICATOR:
            String rowId = uri.getPathSegments().get(1);
            count = db.update(BookTableMetaData.TABLE_NAME,
                Values, BookTableMetaData._ID + "=" + rowId
     + (!TextUtils.isEmpty(where) ? " AND (" + where + ')' : ""),
                whereArgs);
            break;
        default:
            throw new IllegalArgumentException(
                "Nieznany ident. URI " + uri);
        }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```

Przykład – dostawca treści

```
public int delete(Uri uri, String where, String[] whereArgs) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri)) {
        case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
            count = db.delete(BookTableMetaData.TABLE_NAME,
                              where, whereArgs);
            break;
        case INCOMING_SINGLE_BOOK_URI_INDICATOR:
            String rowId = uri.getPathSegments().get(1);
            count = db.delete(BookTableMetaData.TABLE_NAME,
                BookTableMetaData._ID + "=" + rowId
    + (!TextUtils.isEmpty(where) ? " AND (" + where + ')' : ""),
                whereArgs);
            break;
        default:
            throw new IllegalArgumentException(
                "Nieznany ident. URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```