



Systemy operacyjne na platformach mobilnych

Wykład 5

*Grzegorz Jabłoński, Piotr Perek
Katedra Mikroelektroniki i Technik Informatycznych*

Zagadnienia wykładu

- Przechowywanie danych
 - Preferencje
 - System plików: pamięć wewnętrzna i karta SD
 - Baza danych SQLite

Przechowywanie danych

Android udostępnia kilka metod przechowywania danych:

- System plików – pamięć wewnętrzna (aplikacji) oraz zewnętrzna (karta SD)
- Lokalna baza danych (SQLite)
- Preferencje – umożliwiają przechowywanie niewielkich ilości danych w postaci par klucz-wartość

Preferencje

- System preferencji umożliwia przechowywanie niewielkich paczek informacji w postaci par klucz-wartość
- Może być stosowany do przechowania ustawień aplikacji oraz przekazywania informacji pomiędzy aktywnościami
- Przechowywane dane mogą być oznaczone jako prywatne lub dostępne dla innych aplikacji

Preferencje

- Interfejs `SharedPreferences` dostarcza zestaw metod umożliwiających odczyt i modyfikację danych
- Dostęp do obiektu `SharedPreferences` można uzyskać z wykorzystaniem metody `getSharedPreferences` klasy `Context`

```
getSharedPreferences (String name, int accessMode)
```

- `name` – plik wykorzystywany przez właściwości, jeśli plik nie istnieje jest tworzony
- `accessMode` – tryb dostępu

Preferencje – tryby dostępu

- Tryb dostępu umożliwia ustawienie uprawnień do odczytu/zapisu dla pliku preferencji
- Wspierane tryby:
 - `Context.MODE_PRIVATE` – dostęp tylko dla aplikacji, która utworzyła plik
 - `Context.MODE_WORLD_READABLE` – dostęp do odczytu dla innych aplikacji
 - `Context.MODE_WORLD_WRITEABLE` – dostęp do zapisu dla innych aplikacji

Preferencje - przykład

```
public class SharedPrefTestInput extends Activity {
public static final String PREFS_PRIVATE = "PREFS_PRIVATE";
public static final String PREFS_WORLD_READ = "PREFS_WORLD_READABLE";
public static final String PREFS_WORLD_WRITE = "PREFS_WORLD_WRITABLE";
public static final String PREFS_WORLD_READ_WRITE =
                                "PREFS_WORLD_READABLE_WRITABLE";
public static final String KEY_PRIVATE = "KEY_PRIVATE";
public static final String KEY_WORLD_READ = "KEY_WORLD_READ";
public static final String KEY_WORLD_WRITE = "KEY_WORLD_WRITE";
public static final String KEY_WORLD_READ_WRITE =
                                "KEY_WORLD_READ_WRITE";
. . .

private SharedPreferences prefsPrivate;
private SharedPreferences prefsWorldRead;
private SharedPreferences prefsWorldWrite;
private SharedPreferences prefsWorldReadWrite;
```

Preferencje - przykład

```
public void onCreate(Bundle savedInstanceState) {  
    .....  
    prefsPrivate = getSharedPreferences(  
        SharedPrefTestInput.PREFS_PRIVATE,  
        Context.MODE_PRIVATE);  
    prefsWorldRead = getSharedPreferences(  
        SharedPrefTestInput.PREFS_WORLD_READ,  
        Context.MODE_WORLD_READABLE);  
    prefsWorldWrite = getSharedPreferences(  
        SharedPrefTestInput.PREFS_WORLD_WRITE,  
        Context.MODE_WORLD_WRITEABLE);  
    prefsWorldReadWrite = getSharedPreferences(  
        SharedPrefTestInput.PREFS_WORLD_READ_WRITE,  
        Context.MODE_WORLD_READABLE  
        + Context.MODE_WORLD_WRITEABLE);  
}
```


Preferencje – edytor

- Do modyfikacji preferencji należy użyć obiektu klasy Editor
- Aby utworzyć obiekt Editor, który umożliwia modyfikację pliku preferencji należy użyć metody edit() z klasy SharedPreferences
- Edytor umożliwia dodawanie preferencji typów: String, boolean, float, int oraz long
- Aby zmiany wprowadzone przy pomocy edytora zostały zapisane w pliku preferencji należy użyć metody commit()

Preferencje - zapis

```
Editor prefsPrivateEditor = prefsPrivate.edit();
Editor prefsWorldReadEditor = prefsWorldRead.edit();
Editor prefsWorldWriteEditor = prefsWorldWrite.edit();
Editor prefsWorldReadWriteEditor = prefsWorldReadWrite.edit();

prefsPrivateEditor.putString(SharedPrefTestInput.KEY_PRIVATE,
                             inputPrivate.getText().toString());
prefsWorldReadEditor.putString(SharedPrefTestInput.KEY_WORLD_READ,
                               inputWorldRead.getText().toString());
prefsWorldWriteEditor.putString(SharedPrefTestInput.KEY_WORLD_WRITE,
                                inputWorldWrite.getText().toString());
prefsWorldReadWriteEditor.putString(SharedPrefTestInput.KEY_WORLD_READ_WRITE,
                                    inputWorldReadWrite.getText().toString());

prefsPrivateEditor.commit();
prefsWorldReadEditor.commit();
prefsWorldWriteEditor.commit();
prefsWorldReadWriteEditor.commit();
```

Preferencje - odczyt

```
outputPrivate.setText (prefsPrivate.getString (
    SharedPrefTestInput.KEY_PRIVATE, "NA"));
outputWorldRead.setText (prefsWorldRead.getString (
    SharedPrefTestInput.KEY_WORLD_READ, "NA"));
outputWorldWrite.setText (prefsWorldWrite.getString (
    SharedPrefTestInput.KEY_WORLD_WRITE, "NA"));
outputWorldReadWrite.setText (prefsWorldReadWrite.getString (
    SharedPrefTestInput.KEY_WORLD_READ_WRITE, "NA"));
```

System plików

- System plików Androida bazuje na Linuksie i wspiera definiowanie trybów odczytu zapisu dla poszczególnych plików
- Dostęp do plików może być realizowany na różne sposoby:
 - odczyt/zapis plików z katalogu aplikacji
 - odczyt „surowych” plików zapisanych jako zasoby aplikacji
 - odczyt plików XML zapisanych jako zasoby aplikacji

Zapis/odczyt plików

– katalog prywatny aplikacji

- Android udostępnia proste metody umożliwiające zapis/odczyt plików tworzonych w katalogu prywatnym aplikacji
- Metoda `openFileOutput(String name, int mode)`, zwraca strumień wyjściowy do pliku o nazwie `name`. Jeśli plik nie istnieje jest tworzony.
- Metoda `openFileInput(String name)`, zwraca strumień wejściowy z pliku o nazwie `name`.
- Pliki przechowywane są w katalogu:
`data/data/[PACKAGE_NAME]/files/`

Zapis pliku - przykład

```
FileOutputStream fos = null;
try {
    fos = openFileOutput("filename.txt", Context.MODE_PRIVATE);
    fos.write(createInput.getText().toString().getBytes());
} catch (FileNotFoundException e) {
    Log.e("CreateFile", e.getLocalizedMessage());
} catch (IOException e) {
    Log.e("CreateFile", e.getLocalizedMessage());
} finally {
    if (fos != null) {
        try {
            fos.flush();
            fos.close();
        } catch (IOException e) {
            // swallow
        }
    }
}
```

Odczyt pliku - przykład

```
FileInputStream fis = null;
try {
    fis = openFileInput("filename.txt");
    byte[] reader = new byte[fis.available()];
    while (fis.read(reader) != -1) {}
    readOutput.setText(new String(reader));
} catch (IOException e) {
    Log.e("ReadFile", e.getMessage(), e);
} finally {
    if (fis != null) {
        try {
            fis.close();
        } catch (IOException e) {
            // swallow
        }
    }
}
```

Pliki zasobów

- Android umożliwia dołączenie plików do aplikacji w postaci zasobów
- Jeśli plik zostanie umieszczony w katalogu `res/raw` nie jest w żaden sposób przetwarzany, ale jest udostępniany jako zasób

Pliki zasobów - przykład

```
Resources resources = getResources();
InputStream is = null;
try {
    is = resources.openRawResource(R.raw.people);
    byte[] reader = new byte[is.available()];
    while (is.read(reader) != -1) {}
    readOutput.setText(new String(reader));
} catch (IOException e) {
    Log.e("ReadRawResourceFile", e.getMessage(), e);
} finally {
    if (is != null) {
        try {
            is.close();
        } catch (IOException e) {
            // swallow
        }
    }
}
```

Pliki XML - zasoby

- Odczyt plików XML zapisanych w katalogu `res/xml` różni się od odczytu „surowych” plików zapisanych w katalogu `res/raw` ponieważ są one kompilowane do postaci binarnej
- Dostęp do danych zapisanych w pliku XML odbywa się za pomocą parsera `XmlPullParser`

Pliki XML - przykład

```
XmlPullParser parser = getResources().getXml(R.xml.people);
StringBuffer sb = new StringBuffer();
try {
    while (parser.next() != XmlPullParser.END_DOCUMENT) {
        String name = parser.getName();
        String first = null, last = null;
        if ((name != null) && name.equals("person")) {
            int size = parser.getAttributeCount();
            for (int i = 0; i < size; i++) {
                String attrName = parser.getAttributeName(i);
                String attrValue = parser.getAttributeValue(i);
                if ((attrName != null) && attrName.equals("firstname")) {
                    first = attrValue;
                } else if ((attrName != null) && attrName.equals("lastname")) {
                    last = attrValue;
                }
            }
            if ((first != null) && (last != null)) {
                sb.append(last + ", " + first + "\n");
            }
        }
    }
    readOutput.setText(sb.toString());
} catch (Exception e) {
    Log.e("ReadXMLResourceFile", e.getMessage(), e);
}
```

Pamięć zewnętrzna – karta SD

- Android dostarcza również prosty interfejs umożliwiający dostęp do karty SD zamontowanej w urządzeniu
- Karta SD powinno być używana do przechowywania plików o dużym rozmiarze np. obrazów, filmów itp.
- W przypadku karty SD nie istnieją tryby dostępu i uprawnienia jak w systemie Linux
- Każda aplikacja ma prawo odczytu danych zapisanych na karcie SD, jednak aby mieć możliwość zapisu należy zdefiniować uprawnienie w pliku AndroidManifest.xml

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

Dostęp do karty SD - przykład

```
File sdDir = Environment.getExternalStorageDirectory();
if (sdDir.exists() && sdDir.canWrite()) {
    File uadDir = new File(sdDir.getAbsolutePath() + "/unlocking_android");
    uadDir.mkdir();
    if (uadDir.exists() && uadDir.canWrite()) {
        File file = new File(uadDir.getAbsolutePath() + "/" + fileName);
        try {
            file.createNewFile();
        } catch (IOException e) {
            // log and or handle
        }
        if (file.exists() && file.canWrite()) {
            FileOutputStream fos = null;
            try {
                fos = new FileOutputStream(file);
                fos.write("I fear you speak upon the rack,"
                    + "where men enforced do speak "
                    + "anything.".getBytes());
            }
        }
    }
}
. . .
```

Baza danych SQLite

- SQLite – baza danych typu open source
- Jest dostępna na każdym urządzeniu z systemem Android
- Nie wymaga konfiguracji, ani administrowania
- Wspiera standard relacyjnych baz danych:
 - składnia SQL
 - transakcje
 - zapytania parametryzowane
- Ma małe wymagania pamięciowe (w przybliżeniu ok. 250 kB)

Baza danych SQLite

- Typy danych wspierane przez bazę SQLite:
 - TEXT (podobny do typu String z Javy)
 - INTEGER (podobny do typu long z Javy)
 - REAL (podobny do typu double z Javy)
- Wszystkie pozostałe typy należy skonwertować do powyższych przed ich zapisem do bazy
- SQLite nie waliduje typów danych wpisywanych do kolumn (można np. wpisać liczbę do kolumny tekstowej i na odwrót)
- klucz główny tabeli musi być typu INTEGER i posiadać nazwę „_id”

Baza danych SQLite

- Dostęp do bazy SQLite wymaga dostępu do systemu plików
- Dostęp ten może być powolny – zaleca się wykonywanie asynchronicznie operacji na bazie (np. z użyciem klasy `AsyncTask`)
- Aplikacje tworzą pliki bazy danych w katalogu: *DATA/data/APP_NAME/databases/FILENAME*
- gdzie:
 - DATA - ścieżka zwracana przez metodę: `Environment.getDataDirectory()`
 - APP_NAME - nazwa aplikacji
 - FILENAME - nazwa bazy danych

Baza danych SQLite

- Pakiet `android.database` zawiera wszystkie podstawowe klasy potrzebne przy pracy z bazami danych
- Pakiet `android.database.sqlite` zawiera klasy dedykowane dla bazy SQLite
- Do utworzenia i aktualizacji bazy danych najczęściej wykorzystuje się klasę rozszerzającą abstrakcyjną klasę `android.database.sqlite.SQLiteOpenHelper`
- Konstruktor klasy:

```
public SQLiteOpenHelper(Context context, String name,  
                          SQLiteDatabase.CursorFactory factory, int version)
```

SQLiteOpenHelper

- W klasie helpera należy zdefiniować metody:

```
public abstract void onCreate(SQLiteDatabase db)
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion,
                               int newVersion)
```

- Metoda onCreate jest wywołana, o ile baza nie istnieje
- Metoda onUpgrade jest wołana, gdy nastąpi zmiana schematu bazy

SQLiteOpenHelper

- Klasa udostępnia metody dające dostęp do bazy w trybie do odczytu oraz do odczytu/zapisu

```
public synchronized SQLiteDatabase getReadableDatabase()  
public synchronized SQLiteDatabase getWritableDatabase()
```

SQLiteDatabase

- Klasa `android.database.sqlite.SQLiteDatabase` definiuje metody umożliwiające zarządzanie bazą SQLite, m.in.:

```
public long insert(String table, String nullColumnHack, ContentValues values)
public int update(String table, ContentValues values, String whereClause,
                  String[] whereArgs)
public int delete(String table, String whereClause, String[] whereArgs)
public void execSQL(String sql)
```

SQLiteDatabase

- Metody zapytań

```
public Cursor query(String table, String[] columns, String selection,  
                    String[] selectionArgs, String groupBy, String having,  
                    String orderBy, String limit)  
public Cursor query(String table, String[] columns, String selection,  
                    String[] selectionArgs, String groupBy, String having,  
                    String orderBy)  
public Cursor query(boolean distinct, String table, String[] columns,  
                    String selection, String[] selectionArgs, String groupBy,  
                    String having, String orderBy, String limit)  
public Cursor.rawQuery(String sql, String[] selectionArgs)
```

Baza danych SQL - przykład

```
public class Comment {  
    private long id;  
    Private String comment;  
  
    @Override  
    public String toString() {  
        return comment;  
    }  
}
```

Baza danych SQL - przykład

```
public class DbHelper extends SQLiteOpenHelper {
    private static final String TAG = DbHelper.class.getName();
    public static final String TABLE_NAME = "comments";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_COMMENT = "comment";

    private static final String DATABASE_NAME = "comments.db";
    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_CREATE = "create table " + TABLE_NAME
        + "( " + COLUMN_ID + " integer primary key autoincrement, "
        + COLUMN_COMMENT + " text not null)";
    private static final String DATABASE_DROP= "DROP TABLE IF EXISTS "
        + TABLE_NAME;
```

Baza danych SQL - przykład

```
public class DBHelper extends SQLiteOpenHelper
{
    . . .
    public DBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_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(DATABASE_DROP);
        onCreate(db);
    }
    . . .
}
```


Baza danych SQL - przykład

```
public class DBHelper extends SQLiteOpenHelper
{
    . . .
    public DBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_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(DATABASE_DROP);
        onCreate(db);
    }
    . . .
}
```

Baza danych SQL - przykład

```
public class CommentsDataSource {
    private final static String TAG = CommentsDataSource.class.getName();
    private SQLiteDatabase db;
    private DbHelper dbHelper;
    private String[] allColumns = { COLUMN_ID, COLUMN_COMMENT };

    public CommentsDataSource(Context context) {
        dbHelper = new DbHelper(context);
    }

    public void open() throws SQLException {
        db = dbHelper.getWritableDatabase();
    }

    public void close() {
        dbHelper.close();
    }
    . . .
}
```

Baza danych SQL - przykład

```
public class CommentsDataSource {  
    . . .  
    private Comment cursorToComment(Cursor cursor) {  
        Comment comment = new Comment();  
        comment.setId(cursor.getLong(0));  
        comment.setComment(cursor.getString(1));  
        return comment;  
    }  
    public Comment createComment(String comment) {  
        ContentValues values = new ContentValues();  
        values.put(COLUMN_COMMENT, comment);  
        longid = db.insert(TABLE_NAME, null, values);  
        Cursor cursor = db.query(TABLE_NAME, allColumns, COLUMN_ID + " = " + id,  
                                null, null, null, null);  
  
        cursor.moveToFirst();  
        Comment newComment = cursorToComment(cursor);  
        cursor.close();  
        Log.i(TAG, "Comment inserted with id: " + id);  
        return newComment;  
    }  
}
```

Baza danych SQL - przykład

```
public class CommentsDataSource {  
    . . .  
    public void deleteComment(Comment comment) {  
        longid = comment.getId();  
        Log.i(TAG, "Comment deleted with id: " + id);  
        db.delete(TABLE_NAME, COLUMN_ID + " = " + id, null);  
    }  
  
    public List<Comment> getAllComments() {  
        List<Comment> comments = new ArrayList<Comment>();  
        Cursor cursor = db.query(TABLE_NAME, allColumns, null, null, null,  
                                null, null);  
  
        cursor.moveToFirst();  
        while(!cursor.isAfterLast()) {  
            Comment comment = cursorToComment(cursor);  
            comments.add(comment);  
            cursor.moveToNext();  
        }  
        cursor.close();  
        return comments;  
    }  
}
```