**ЛЕКЦІЯ 16. РОБОТА З ФАЙЛОВОЮ СИСТЕМОЮ**

**11.1 Читання і збереження файлів**

Робота з налаштуваннями рівня *activity* і застосунку дозволяє зберегти невеликі дані окремих типів (*string*, *int*), але для роботи з великими масивами даних, такими як графічні файли, файли мультимедіа і т. д., нам доведеться звертатися до файлової системи.

ОС Android побудована на основі Linux. Цей факт знаходить своє відображення в роботі з файлами. Так, в шляхах до файлів, в якості розділювача, Linux використовує слеш *"/"*, а не зворотний слеш *"\"* (як у Windows). А всі назви файлів і каталогів є чутливими до регістру, тобто *"data"* це не те ж саме, що і *"Data"*.

Додаток Android зберігає свої дані в каталозі */data/data/<назва\_пакета>/* і, як правило, відносно цього каталогу буде відбуватися робота.

Для роботи з файлами абстрактний клас *android.content.Context* визначає ряд методів:

● *deleteFile(String name)*: видаляє певний файл;

● *fileList*(): повертає всі файли, які містяться в підкаталозі /*files* каталога додатку;

● *getCacheDir*(): повертає посилання на підкаталог /*cache* каталога додатку;

● *getDir(String dirName, int mode)*: повертає посилання на підкаталог в каталозі додатки, якщо такого підкаталогу немає, то він створюється;

● *getExternalCacheDir*(): повертає посилання на папку /*cache* зовнішньої файлової системи пристрою;

● *getExternalFilesDir*(): повертає посилання на каталог /*files* зовнішньої файлової системи пристрою;

● *getFileStreamPath(String filename)*: повертає абсолютний шлях до файлу в файловій системі;

● *openFileInput(String filename)*: відкриває файл для читання;

● *openFileOutput (String name, int mode)*: відкриває файл для запису.

Всі файли, які створюються і редагуються в програмі, як правило, зберігаються в підкаталозі */files* в каталозі застосунку.

Для безпосереднього читання і запису файлів застосовуються також стандартні класи Java з пакету *java.io*.

Отже, можна застосувати функціонал читання-запису файлів в додатку. Нехай у нас буде наступна примітивна розмітка layout:

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:layout\_width="match\_parent" android:layout\_height="match\_parent"

android:orientation="vertical">

<EditText android:id=["@+id](mailto:@+id)/save\_text" android:layout\_width="match\_parent" android:layout\_height="0dp" android:gravity="start" android:layout\_weight="4"/>

<Button android:layout\_width="wrap\_content" android:layout\_height="0dp" android:layout\_weight="1" android:layout\_margin="16dp" android:layout\_gravity="center" android:onClick="saveText" android:text="Сохранить"/>

<TextView android:layout\_marginTop="80dp" android:id=["@+id](mailto:@+id)/open\_text" android:layout\_width="match\_parent" android:layout\_height="0dp" android:gravity="start" android:layout\_weight="4"/>

<Button android:layout\_width="wrap\_content" android:layout\_height="0dp" android:layout\_margin="16dp" android:layout\_weight="1" android:layout\_gravity="center" android:onClick="openText"

android:text="Открыть"/>

</LinearLayout>

Поле *EditText* призначене для введення тексту, а *TextView* – для виведення попередньо записаного тексту. Для збереження і відновлення тексту додані дві кнопки.

Тепер в коді *Activity* пропишемо обробники кнопок зі збереженням і читанням

файлу:

public class MainActivity extends AppCompatActivity {

private final static String FILE\_NAME="content.txt";

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

// збереження файла

public void saveText(View view){ FileOutputStream fos=null;

try {

EditText textBox=(EditText) findViewById(R.id.save\_text); String text=textBox.getText().toString(); fos=openFileOutput(FILE\_NAME, MODE\_PRIVATE); fos.write(text.getBytes());

Toast.makeText(this, "Файл сохранен", Toast.LENGTH\_SHORT).show();

}

catch(IOException ex) {

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

finally{

try{

if(fos!=null)

fos.close();

}

catch(IOException ex){

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

}

}

// відкриття файла

public void openText(View view){ FileInputStream fin = null;

TextView textView = (TextView) findViewById(R.id.open\_text);

try { fin=openFileInput(FILE\_NAME); byte[] bytes=new byte[fin.available()]; fin.read(bytes);

String text=new String (bytes);

textView.setText(text);

}

catch(IOException ex) {

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

finally{

try{

if(fin!=null)

fin.close();

}

catch(IOException ex){

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

}

}

}

При натисканні на кнопку збереження буде створюватися потік виведення

FileOutputStream fos=openFileOutput(FILE\_NAME, MODE\_PRIVATE).

В даному випадку введений текст буде зберігатися в файл *"content.txt"*. При цьому буде використовуватися режим **MODE\_PRIVATE**.

Система дозволяє створювати файли з двома різними режимами:

● **MODE\_PRIVATE** : файли можуть бути доступні тільки власнику додатки

(режим за замовчуванням);

● **MODE\_APPEND** : дані можуть бути додані в кінець файлую

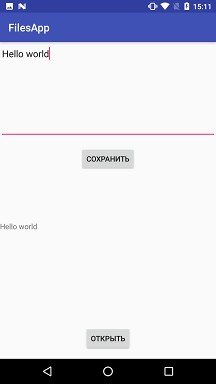
Тому в даному випадку якщо файл *"content.txt"* вже існує, то він буде перезаписаний. Якщо ж нам треба було дописати файл, тоді треба було б використовувати режим **MODE\_APPEND**:

FileOutputStream fos=openFileOutput(FILE\_NAME, MODE\_APPEND);

Для читання файлу застосовується потік введення *FileInputStream* : FileInputStream fin=openFileInput(FILE\_NAME);

Детальніше про використання потоків введення-виведення можна прочитати в будь-якій документації по Java.

У підсумку після натискання на кнопку збереження весь текст буде збережений у файлі */data/data/назва\_пакету/files/content.txt*:



**11.2 Розміщення файлів у зовнішньому сховищі**

У попередньому підрозділі ми розглянули збереження і читання файлів з каталогу програми. За замовчуванням такі файли доступні тільки самому застосунку. Однак ми можемо поміщати і працювати з файлами із зовнішнього сховища. Це також дозволить іншим програмам відкривати дані файли і при необхідності змінювати.

Весь механізм роботи з файлами буде таким же, як і при роботі зі сховищем застосунку. Ключовою відмінністю тут буде отримання і використання шляху до зовнішнього сховища через метод *Environment.getExternalStorageDirectory().*

Отже, нехай у файлі *activity\_main.xml* буде така ж розмітка інтерфейсу:

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:layout\_width="match\_parent" android:layout\_height="match\_parent"

android:orientation="vertical">

<EditText android:id=["@+id](mailto:@+id)/save\_text" android:layout\_width="match\_parent" android:layout\_height="0dp" android:gravity="start" android:layout\_weight="4"/>

<Button android:layout\_width="wrap\_content" android:layout\_height="0dp" android:layout\_weight="1" android:layout\_margin="16dp" android:layout\_gravity="center" android:onClick="saveText" android:text="Сохранить"/>

<TextView android:layout\_marginTop="80dp" android:id=["@+id](mailto:@+id)/open\_text" android:layout\_width="match\_parent" android:layout\_height="0dp" android:gravity="start" android:layout\_weight="4"/>

<Button android:layout\_width="wrap\_content" android:layout\_height="0dp" android:layout\_margin="16dp" android:layout\_weight="1"

android:layout\_gravity="center" android:onClick="openText" android:text="Открыть"/>

</LinearLayout>

А код класу *MainActivity* буде виглядати наступним чином:

public class MainActivity extends AppCompatActivity {

private final static String FILE\_NAME = "content.txt";

private static final int REQUEST\_PERMISSION\_WRITE = 1001;

private boolean permissionGranted;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

private File getExternalPath() {

return(new File(Environment.getExternalStorageDirectory(), FILE\_NAME));

}

// збереження файла

public void saveText(View view){

if(!permissionGranted){

checkPermissions();

return;

}

FileOutputStream fos = null;

try {

EditText textBox = (EditText) findViewById(R.id.save\_text); String text = textBox.getText().toString();

fos = new FileOutputStream(getExternalPath());

fos.write(text.getBytes());

Toast.makeText(this, "Файл сохранен", Toast.LENGTH\_SHORT).show();

}

catch(IOException ex) {

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

finally{

try{

if(fos!=null)

fos.close();

}

catch(IOException ex){

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

}

}

// відкриття файла

public void openText(View view){

if(!permissionGranted){

checkPermissions();

return;

}

FileInputStream fin = null;

TextView textView = (TextView) findViewById(R.id.open\_text); File file = getExternalPath();

// якщо файл не існує, відбувається вихід из коду метода

if(!file.exists()) return;

try {

fin = new FileInputStream(file);

byte[] bytes = new byte[fin.available()];

fin.read(bytes);

String text = new String (bytes);

textView.setText(text);

}

catch(IOException ex) {

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

finally{

try{

if(fin!=null)

fin.close();

}

catch(IOException ex){

Toast.makeText(this, ex.getMessage(), Toast.LENGTH\_SHORT).show();

}

}

}

// перевіряємо, чи є доступним зовнішнє сховище для читання і запису

public boolean isExternalStorageWriteable(){

String state = Environment.getExternalStorageState();

return Environment.MEDIA\_MOUNTED.equals(state);

}

// перевіряємо, чи є доступним зовнішнє сховище хоча б для читання

public boolean isExternalStorageReadable(){

String state = Environment.getExternalStorageState();

return (Environment.MEDIA\_MOUNTED.equals(state) || Environment.MEDIA\_MOUNTED\_READ\_ONLY.equals(state));

}

private boolean checkPermissions(){

if(!isExternalStorageReadable() || !isExternalStorageWriteable()){ Toast.makeText(this, "Зовнішнє сховище не доступне",

Toast.LENGTH\_LONG).show();

return false;

}

int permissionCheck = ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE\_EXTERNAL\_STORAGE);

if(permissionCheck!= PackageManager.PERMISSION\_GRANTED){ ActivityCompat.requestPermissions(this, new

String[]{Manifest.permission.WRITE\_EXTERNAL\_STORAGE}, REQUEST\_PERMISSION\_WRITE);

return false;

}

return true;

}

[@Override](mailto:@Override)

public void onRequestPermissionsResult(int requestCode[, @NonNull](mailto:@NonNull) String[] permissions,

[@NonNull](mailto:@NonNull) int[] grantResults){

switch (requestCode){

case REQUEST\_PERMISSION\_WRITE:

if(grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED){

permissionGranted=true;

Toast.makeText(this, "Дозвіл отримано", Toast.LENGTH\_LONG).show();

}

else{

Toast.makeText(this, "Необхідно отримати дозвіл", Toast.LENGTH\_LONG).show();

}

break;

}

}

}

За допомогою виразу *Environment.getExternalStorageDirectory()*, отримуємо

доступ до папки програми в зовнішньому сховищі і встановлюємо об’єкт файлу:

private File getExternalPath() {

return(new File(Environment.getExternalStorageDirectory(), FILE\_NAME));

}

Оскільки для читання/запису в зовнішнє сховище необхідні дозволи, то перед

операціями збереження та записи файлу необхідно перевірити наявність дозволів. Для цього визначено метод *checkPermissions*(). При установці дозволів спрацьовує метод *onRequestPermissionsResult*(), в якому в разі вдалої установки дозволів для змінної *permissionGranted* задається значення *true*.

Щоб використовувати зовнішнє сховище, також треба встановити дозволу в файлі маніфесту *AndroidManifest.xml*:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="[http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android)

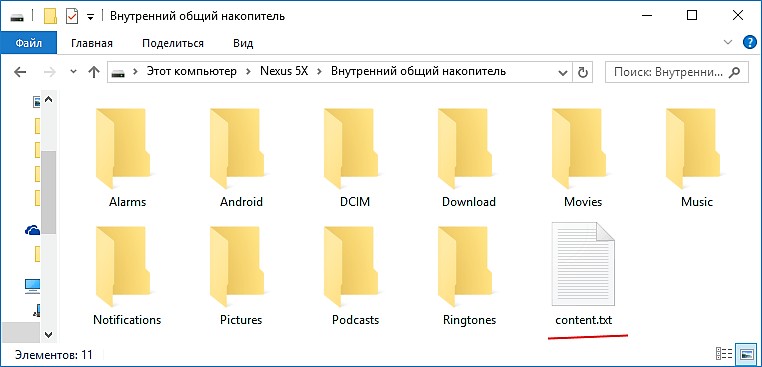
package="com.example.eugene.filesapp" >

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

//............................

І після операції запису на смартфоні ми зможемо побачити створений файл:



**11.3 Робота з JSON**

Для роботи з форматом *json* немає вбудованих засобів, але є багато бібліотек і пакетів. Одним із найбільш популярних з них є пакет *com.google.code.gson* .

Для його використання в проекті Android, необхідно додати відповідну залежність в файл *guild.gradle*, який відноситься до модуля *app*:

compile 'com.google.code.gson:gson:2.8.0'

Тобто після додавання, секція залежностей в файлі *build.gradle* може виглядати наступним чином:

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {

exclude group: 'com.android.support', module: 'support-annotations'

})

compile 'com.android.support:appcompat-v7:25.3.0'

compile 'com.android.support.constraint:constraint-layout:1.0.2'

compile 'com.google.code.gson:gson:2.8.0' testCompile 'junit:junit:4.12'

}

Після додавання пакета в проект, додамо новий клас *Phone*, який буде

представляти дані:

package com.example.eugene.jsonapp;

public class Phone { private String name; private int price;

Phone(String name, int price){

this.name=name;

this.price=price;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name=name;

}

public int getPrice() {

return price;

}

public void setPrice(int price) {

this.price=price;

}

[@Override](mailto:@Override)

public String toString(){

return name+" "+String.valueOf(price);

}

}

Об’єкти цього класу ми будемо серіалізовувати у формат *json* і навпаки

десеріалізовувати з файлу.

Для роботи з *json* додамо наступний клас *JSONHelper*:

class JSONHelper {

private static final String FILE\_NAME="data.json";

static boolean exportToJSON(Context context, List<Phone> dataList) { Gson gson=new Gson();

DataItems dataItems=new DataItems();

dataItems.setPhones(dataList);

String jsonString=gson.toJson(dataItems); FileOutputStream fileOutputStream=null; try {

fileOutputStream = context.openFileOutput(FILE\_NAME, Context.MODE\_PRIVATE);

fileOutputStream.write(jsonString.getBytes());

return true;

} catch (Exception e) {

e.printStackTrace();

} finally {

if (fileOutputStream !=null) {

try {

fileOutputStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

return false;

}

static List<Phone> importFromJSON(Context context) { InputStreamReader streamReader=null; FileInputStream fileInputStream=null;

try{ fileInputStream=context.openFileInput(FILE\_NAME); streamReader=new InputStreamReader(fileInputStream); Gson gson=new Gson();

DataItems dataItems=gson.fromJson(streamReader, DataItems.class);

return dataItems.getPhones();

}

catch (IOException ex){

ex.printStackTrace();

}

finally {

if (streamReader !=null) {

try {

streamReader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if (fileInputStream !=null) {

try {

fileInputStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

return null;

}

private static class DataItems {

private List<Phone>phones;

List<Phone> getPhones() {

return phones;

}

void setPhones(List<Phone> phones) {

this.phones=phones;

}

}

}

Тут для роботи із *json* створюється об’єкт *Gson* . Для серіалізації даних в формат

*json* у цього об’єкта викликається метод toJson(), в який передаються серіалізовувані дані.

Для спрощення роботи з даними застосовується допоміжний клас *DataItems*. На виході метод *toJson*() повертає рядок, яка потім зберігається в текстовий файл.

Для десеріалізації виконується метод *fromJson*(), в який передається об’єкт

*Reader* з серіалізованими даними і тип, до якого треба десеріалізувати дані.

Тепер визначимо основний функціонал для взаємодії з користувачем. Змінимо файл *activity\_main.xml* наступним чином:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:layout\_width="match\_parent" android:layout\_height="match\_parent"

android:orientation="vertical">

<EditText android:id=["@+id](mailto:@+id)/nameText" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:hint="Введите название"/>

<EditText android:id=["@+id](mailto:@+id)/priceText" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:hint="Введите цену"/>

<Button android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" android:text="Добавить" android:onClick="addPhone"/>

<LinearLayout android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:orientation="horizontal">

<Button android:layout\_width="0dp" android:layout\_weight="1" android:layout\_height="wrap\_content" android:text="Сохранить" android:onClick="save"/>

<Button android:layout\_width="0dp" android:layout\_weight="1" android:layout\_height="wrap\_content" android:text="Открыть" android:onClick="open"/>

</LinearLayout>

<ListView android:id=["@+id](mailto:@+id)/list" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" />

</LinearLayout>

Тут визначені два текстових поля для введення назви моделі і ціни об’єкта *Phone* і одна кнопка для додавання даних в список. Ще одна кнопка виконує серіалізацію даних зі списку в файл, а третя кнопка – відновлення даних з файлу. Для виведення самих даних визначено елемент *ListView*. І змінимо відповідним чином клас *MainActivity*:

public class MainActivity extends AppCompatActivity {

private ArrayAdapter<Phone> adapter; private EditText nameText, priceText; private List<Phone> phones;

ListView listView;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main); nameText=(EditText) findViewById(R.id.nameText); priceText=(EditText) findViewById(R.id.priceText); phones=new ArrayList<>();

listView=(ListView) findViewById(R.id.list);

adapter=new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, phones);

listView.setAdapter(adapter);

}

public void addPhone(View view){

String name=nameText.getText().toString();

int price=Integer.parseInt(priceText.getText().toString()); Phone phone=new Phone(name, price); phones.add(phone);

adapter.notifyDataSetChanged();

}

public void save(View view){

boolean result = JSONHelper.exportToJSON(this, phones);

if(result){

Toast.makeText(this, "Данные сохранены", Toast.LENGTH\_LONG).show();

}

else{

Toast.makeText(this, "Не удалось сохранить данные",

Toast.LENGTH\_LONG).show();

}

}

public void open(View view){

phones = JSONHelper.importFromJSON(this);

if(phones!=null){

adapter=new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, phones);

listView.setAdapter(adapter);

Toast.makeText(this, "Данные восстановлены", Toast.LENGTH\_LONG).show();

}

else{

Toast.makeText(this, "Не удалось открыть данные",

Toast.LENGTH\_LONG).show();

}

}

}

Всі дані знаходяться в списку *phones*, який представляє об’єкт List<Phone>.

Через адаптер цей список зв’язується з *ListView*.

Для збереження і відновлення даних викликаються раніше описані методи в класі *JSONHelper*. В результаті отримаємо:

