CMSC436: Programming Handheld Systems

Fall 2017
Data Management
Today’s Topics

SharedPreferences
Internal Storage
External Storage
SQLite databases
Shared Preferences

Use when you want to store small amounts of primitive data
Internal Storage

Use when you want to store small to medium amounts of private data
External Storage

Use when you want to store larger amounts of non-private data
Databases

Use when you want to store small to large amounts of private, structured data
SharedPreferences

A persistent map that holds key-value pairs of simple data types

Automatically persisted across application sessions
SharedPreferences

Often used for long-term storage of customizable application data, such as:

- Account name
- Favorite WiFi networks
- User customizations
Activity SharedPreferences

Get a SharedPreferences Object associated with a given Activity

Activity.getSharedPreferences (int mode)

    MODE_PRIVATE is default mode
Named SharedPreferences

Get named SharedPreferences file
Single SharedPreferences object for a given name
Context.getSharedPreferences
(String name, int mode)

name – name of SharedPreferences file
mode – e.g., MODE_PRIVATE
Writing SharedPreferences

Call SharedPreferences.edit()
Returns a SharedPreferences.Editor instance
Writing SharedPreferences

Use SharedPreferences.Editor instance to add values to SharedPreferences
putInt(String key, int value)
putString(String key, String value)
remove(String key)
Writing SharedPreferences

Commit edited values with
SharedPreferences.Editor.commit()
Reading SharedPreferences

Use SharedPreferences methods to read values
getAll()
getBoolean(String key, ...)
getString(String key, ...)
When the user presses the play button, the application displays a random number.
The application keeps track of the highest number seen so far.
DataManagement
SharedPreferences

HighScore: 205

Play
Reset
public class SharedPreferenceReadWriteActivity extends Activity {
    ...
    public void onCreate(Bundle savedInstanceState) {
        final SharedPreferences prefs = getPreferences(MODE_PRIVATE);
        ...
        final Button playButton = findViewById(R.id.play_button);
        playButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                ...
                // Get Stored High Score
                if (val > prefs.getInt(HIGH_SCORE_KEY, 0)) {
                    // Get and edit high score
                    SharedPreferences.Editor editor = prefs.edit();
                    editor.putInt(HIGH_SCORE_KEY, val);
                    editor.apply();
                    ...
                }
            }
        });
    }
}
// Reset Button
final Button resetButton = findViewById(R.id.reset_button);
resetButton.setOnClickListener(new OnClickListener() {

    public void onClick(View v) {

        // Set high score to 0
        SharedPreferences.Editor editor = prefs.edit();
        editor.putInt(HIGH_SCORE_KEY, 0);
        editor.apply();

        ...
PreferenceFragment

A Class that supports displaying & modifying user preferences
DataManagementPreferenceFragment

This application displays a PreferenceFragment, which allows the user to enter and change a persistent user name
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/userPreferenceFragment"
    class="course.examples.datamanagement.preferencefragment.ViewAndUpdatePreferencesActivity$UserPreferenceFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

public static class UserPreferenceFragment extends PreferenceFragment {

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Load the preferences from an XML resource
    addPreferencesFromResource(R.xml.userPrefs);
    // Get the username Preference
    mUserNamePreference = getPreferenceManager().findPreference(USERNAME);
    // Attach a listener to update summary when username changes
    mListener = new OnSharedPreferenceChangeListener() {
        public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
            mUserNamePreference.setSummary(sharedPreferences.getString(USERNAME, "None Set"));
        }
    };
}
// Get SharedPreferences object managed by the PreferenceManager for this Fragment
SharedPreferences prefs = getPreferenceManager().getSharedPreferences();

// Register a listener on the SharedPreferences object
prefs.registerOnSharedPreferenceChangeListener(mListener);

// Invoke callback manually to display the current username
mListener.onSharedPreferenceChanged(prefs, USERNAME);
File

Class that represents a file system entity identified by a pathname
File

Storage areas are classified as internal or external.
Internal memory usually used for smaller, application private data sets.
External memory usually used for larger, non-private data sets.
File API

FileOutputStream openFileOutput (String name, int mode)
    Open private file for writing. Creates the file if it doesn't already exist

FileInputStream openFileInput (String name)
    Open private file for reading

Many others. See documentation.
If a text file does not already exist, application writes text to that text file
Application then reads data from the text file and displays it
Line 1: This is a test of the File Writing API
Line 2: This is a test of the File Writing API
Line 3: This is a test of the File Writing API
private final static String fileName = "TestFile.txt";
public void onCreate(Bundle savedInstanceState) {
    if (!getFileStreamPath(fileName).exists()) {
        try {
            writeFile();
        } catch (FileNotFoundException e) {
            ...
        }
    }
    try {
        readFileAndDisplay(textView);
    } catch (IOException e) {
        ...
    }
}
private void writeFile() throws FileNotFoundException {

    FileOutputStream fos = openFileOutput("fileName", MODE_PRIVATE);
    PrintWriter pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(fos)));

    pw.println("Line 1: This is a test of the File Writing API");
    pw.println("Line 2: This is a test of the File Writing API");
    pw.println("Line 3: This is a test of the File Writing API");

    pw.close();
}

private void readFileAndDisplay(TextView tv) throws IOException {

    FileInputStream fis = openFileInput("fileName");
    BufferedReader br = new BufferedReader(new InputStreamReader(fis));

    String line;
    String sep = System.getProperty("line.separator");

    while (null != (line = br.readLine())) {
        tv.append(line + sep);
    }

    br.close();
}
Using External Memory Files

Removable media may appear/disappear without warning
Using External Memory Files

String Environment.

    getExternalStorageState()

MEDIA_MOUNTED - present & mounted with read/write access

MEDIA_MOUNTED_READ_ONLY - present & mounted with read-only access

MEDIA_REMOVED - not present

Etc.
Using External Memory Files

Permission to write external files

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

Application reads an image file from the resources directory
Copies that file to external storage
Reads image data from the file in external storage
then displays the image
public void onCreate(Bundle savedInstanceState) {
    ...
    if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())) {
        String fileName = "painter.png";
        File outFile = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES), fileName);
        if (!outFile.exists()) copyImageToMemory(outFile);

        ImageView imageview = findViewById(R.id.image);
        imageview.setImageURI(Uri.parse("file://" + outFile.getAbsolutePath()));
    }
}
private void copyImageToMemory(File outFile) {
    try {
        
        BufferedOutputStream os = new BufferedOutputStream(
            new FileOutputStream(outFile));

        BufferedInputStream is = new BufferedInputStream(
            getResources().openRawResource(R.raw.painter));

        copy(is, os);

    } catch (FileNotFoundException e) { ... }
}
private void copy(InputStream is, OutputStream os) {
    final byte[] buf = new byte[1024];
    int numBytes;
    try {
        while (-1 != (numBytes = is.read(buf))) {
            os.write(buf, 0, numBytes);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    finally {
        try {
            is.close();
            os.close();
        } catch (IOException e) {
            ...
        }
    }
}
Cache Files

Temporary files that may be deleted by the system when storage is low

These files are removed when application is uninstalled
Cache Files

File Context.getCacheDir()

Returns absolute path to an application-specific directory that can be used for temporary files.
Saving cache files

Context.getExternalCacheDir()
returns a File representing external storage directory for cache files
SQLite

SQLite provides in-memory database
Designed to operate within a very small footprint (e.g., <300kB)
Implements most of SQL92
Supports ACID transactions
  Atomic, Consistent, Isolated & Durable
Using a Database

Recommended method relies on a helper class called SQLiteOpenHelper
Using a Database

Subclass SQLiteOpenHelper

Call super() from subclass constructor to initialize underlying database
Using a Database

Override onCreate()
Override onUpgrade()
Execute CREATE TABLE commands
Using a Database

Use SQLiteOpenHelper methods to open & return underlying database

Execute operations on underlying database
DataManagementSQLite

Application creates an SQLite database and inserts records, some with errors, into it

When user presses the Fix button, the application deletes, updates and redisplay the corrected database records
DataManagement SQL

1. Frank Sinatra
2. Lady Gaga
3. Jawny Cash
4. Ludwig van Beethoven
public class DatabaseExampleActivity extends ListActivity {
    private DatabaseOpenHelper mDbHelper;
    private SimpleCursorAdapter mAdapter;
    Cursor mCursor;

    public void onCreate(Bundle savedInstanceState) {
        ...
        mDbHelper = new DatabaseOpenHelper(this);
        clearAll();
        insertArtists();
        mCursor = readArtists();
        mAdapter = new SimpleCursorAdapter(this, R.layout.list_layout, mCursor,
            DatabaseOpenHelper.columns, new int[] { R.id._id, R.id.name }, 0);

        setListAdapter(mAdapter);
    }
}
// Delete all records
private void clearAll() {
    mDbHelper.getReadableDatabase()
        .delete(DatabaseOpenHelper.TABLE_NAME, null, null);
}

// Insert several artist records
private void insertArtists() {

    ContentValues values = new ContentValues();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Frank Sinatra");
    mDbHelper.getWritableDatabase()
        .insert(DatabaseOpenHelper.TABLE_NAME, null, values);
    ...
    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Ludwig van Beethoven");
    mDbHelper.getWritableDatabase()
        .insert(DatabaseOpenHelper.TABLE_NAME, null, values);
}
// Returns all artist records in the database
private Cursor readArtists() {
    return mDbHelper.getWritableDatabase().query(
        DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.columns, null, new String[] {}, null, null, null);
}
public void onClick(View v) {

    // Execute database operations
    fix();

    // Redisplay data
    mCursor = readArtists();
    mAdapter.changeCursor(mCursor);
}
private void fix() {

    // Sorry Lady Gaga :-(
    mDbHelper.getWritableDatabase()
        .delete(DatabaseOpenHelper.TABLE_NAME,
                DatabaseOpenHelper.ARTIST_NAME + "=?", new String[] { "Lady Gaga" });

    // fix the Man in Black
    ContentValues values = new ContentValues();
    values.put(DatabaseOpenHelper.ARTIST_NAME, "Johnny Cash");

    mDbHelper.getWritableDatabase()
        .update(DatabaseOpenHelper.TABLE_NAME, values,
                DatabaseOpenHelper.ARTIST_NAME + "=?", new String[] { "Jawny Cash" });
}
Examining the Database Remotely

Databases stored in

/data/data/<package name>/databases/

Can examine database with sqlite3

# adb -s emulator-5554 shell

# sqlite3
/data/data/course.examples.datamanagement.sql/data bases/artist_db
Next Time

Lifecycle-Aware Components