CMSC436: Programming Handheld Systems
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
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 settings
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 SharedPreference 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

DataManagementSharedPreference
HighScore: 205

205

Play
Reset
fun onClickPlayButton(v: View) {

    val highScore = Random().nextInt(1000)
    mGameScore.text = highScore.toString()

    // Get Stored High Score
    if (highScore > mPrefs.getInt(HIGH_SCORE_KEY, 0)) {

        // Get and edit high score
        val editor = mPrefs.edit()
        editor.putInt(HIGH_SCORE_KEY, highScore)
        editor.apply()

        mHighScore.text = highScore.toString()

    }
}

fun onClickResetButton(v: View) {

    // Set high score to 0
    val editor = mPrefs.edit()
    editor.putInt(HIGH_SCORE_KEY, 0)
    editor.apply()

    mHighScore.text = "0"
    mGameScore.text = "0"
}
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
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pref_fragment"
    class="course.examples.datamanagement.preferencefragment.ViewAndUpdatePreferencesActivity$UserPreferenceFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
<android.support.v7.preference.PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">

    <android.support.v7.preference.EditTextPreference
        android:dialogMessage="@string/enter_user_name_string"
        android:dialogTitle="@string/change_user_name_string"
        android:key="uname"
        android:negativeButtonText="@string/cancel_string"
        android:positiveButtonText="@string/submit_string"
        android:title="User Name"/>

</android.support.v7.preference.PreferenceScreen>
UserPreferenceFragment

// Fragment that displays the username preference
class UserPreferenceFragment : PreferenceFragmentCompat() {
    private lateinit var mListener: OnSharedPreferenceChangeListener
    private lateinit var mUserNamePreference: Preference

    override fun onCreatePreferences(p0: Bundle?, p1: String?) {
        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.user_prefs)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Get the username Preference
        mUserNamePreference = preferenceManager.findPreference(USERNAME)
    }
}
UserPreferenceFragment

// Attach a listener to update summary when username changes
mListener = OnSharedPreferenceChangeListener {
    sharedPreferences, _ ->
    mUserNamePreference!!.summary = sharedPreferences.getString(USERNAME, "None Set")
}

// Get SharedPreferences object managed by the PreferenceManager for this Fragment
val prefs = preferenceManager.getSharedPreferences()

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

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

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
DataManagement
FileInternalMemory
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(R.layout.main)
    val textView = findViewById<TextView>(R.id.textview)

    if (!getFileStreamPath(FILE_NAME).exists()) {
        try {
            writeFile()
        } catch (e: FileNotFoundException) {
        }
        try {
            readFileAndDisplay(textView)
        } catch (e: IOException) {
        }
    }
}
@Throws(FileNotFoundException::class)
private fun writeFile() {
    val fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE)
    val pw = PrintWriter(BufferedWriter(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()
}
@Throws(IOException::class)
private fun readFileAndDisplay(tv: TextView) {
    val sep = System.getProperty("line.separator")
    val fis = openFileInput(FILE_NAME)
    val br = BufferedReader(InputStreamReader(fis))

    br.forEachLine {
        tv.append(it + sep)
    }
    br.close()
}
Using External Memory Files

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

String Environment.getExternalStorageState()

Returns

MEDIA_MOUNTED - present & mounted with read/write access

MEDIA_MOUNTED_READ_ONLY - present & mounted with read-only access

MEDIA_REMOVED - not present
Using External Memory Files

Permission to write external files

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

If not done already, application reads an image file from its /res/raw directory

  Copies that file to external storage

Application then reads image data from the file in external storage and then displays the image
DataManagement
FileExternalMemory
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.main)
    if (Environment.MEDIA_MOUNTED == Environment.getExternalStorageState()){
        val fileName = "painter.png"
        val outFile = File(getExternalFilesDir(Environment.DIRECTORY_PICTURES), fileName)

        if (!outFile.exists())
            copyImageToMemory(outFile)

        val imageview = findViewById<ImageView>(R.id.image)
        imageview.setImageURI(Uri.parse("file://" + outFile.absolutePath))
    }
}
private fun copyImageToMemory(outFile: File) {
  var outputStream: BufferedOutputStream? = null
  var inputStream: BufferedInputStream? = null
  try {
    outputStream = BufferedOutputStream(FileOutputStream(outFile))
    inputStream = BufferedInputStream(
      resources.openRawResource(R.raw.painter))
    inputStream.copyTo(outputStream)
  } catch (e: FileNotFoundException) {...
  finally {
    try {
      inputStream?.close()
      outputStream?.close()
    } catch (e: IOException) {...
  }
}
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
Databases

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

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

When user presses the Fix button, the application deletes, updates and redisplays the corrected database records.
DataManagementSQL

1. Frank Sinatra
2. Lady Gaga
3. Johnny Cash
4. Ludwig van Beethoven
```kotlin
public override fun onCreate(savedInstanceState: Bundle?) {
    private lateinit var mDbHelper: DatabaseOpenHelper
    private lateinit var mAdapter: SimpleCursorAdapter
    private var mCursor: Cursor? = null

    mDbHelper = DatabaseOpenHelper(this)
    clearAll()
    insertArtists()
    mCursor = readArtists()
    mAdapter = SimpleCursorAdapter(this, R.layout.list_layout, mCursor,
                                   DatabaseOpenHelper.columns, intArrayOf(R.id._id, R.id.name), 0)

    listAdapter = mAdapter
}
```
DatabaseExampleActivity.kt

// Delete all records
private fun clearAll() {
    mDbHelper.writableDatabase.delete(
        DatabaseOpenHelper.TABLE_NAME, null, null)
}

private fun insertArtists() {
    val values = ContentValues()
    values.put(DatabaseOpenHelper.ARTIST_NAME, "Frank Sinatra")
    mDbHelper.writableDatabase.insert(
            DatabaseOpenHelper.TABLE_NAME, null, values)
    values.clear()
    ...
    values.clear()
    values.put(DatabaseOpenHelper.ARTIST_NAME, "Ludwig van Beethoven")
    mDbHelper.writableDatabase.insert(
            DatabaseOpenHelper.TABLE_NAME, null, values)
}
private fun readArtists(): Cursor {
    return mDbHelper.writableDatabase.query(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.columns, null, arrayOf(), null, null, null)
}
fun onClick(v: View) {

    // Execute database operations
    fix()

    // Redisplay data
    mCursor = readArtists()
    mAdapter.changeCursor(mCursor)
}
private fun fix() {
    // Sorry Lady Gaga :-(
    mDbHelper.writableDatabase.delete(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.ARTIST_NAME + "=?", arrayOf("Lady Gaga")
    )
    // fix the Man in Black
    val values = ContentValues()
    values.put(DatabaseOpenHelper.ARTIST_NAME, "Johnny Cash")

    mDbHelper.writableDatabase.update(DatabaseOpenHelper.TABLE_NAME,
        values, DatabaseOpenHelper.ARTIST_NAME + "=?", arrayOf("Jawny Cash")
    )
}

Examining the Database Remotely

Databases stored in

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

Can examine database with sqlite3

# adb -s emulator-5554 shell

# su

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

Lifecycle-Aware Components
Sample code

DataManagementSharedPreferences
DataManagementPreferenceFragment
DataManagementFileInternalMemory
DataManagementFileExternalMemory
DataManagementSQL