Development with Android

Computing Science and Mathematics
University of Stirling

Data Storage and Exchange

Data Storage Options

- Preferences:
  - a lightweight mechanism to store and retrieve key-value pairs of primitive data types
- File:
  - storing data in files in device memory or in removable storage, e.g. an SD (Secure Data) card
- Database:
  - Android supports SQLite
- Network:
  - use the Internet to store and receive data, whether a relational database or just a text file

Preferences

- Use *Context.getSharedPreferences* to read and write key-value pair
  - Set as private to the app: MODE_PRIVATE
  - Set as public: MODE_WORLD_READABLE

```
SharedPreferences prefs = context.getSharedPreferences("details", MODE_PRIVATE);
Editor edit = prefs.edit(); edit.clear();
edit.putString("username", “abc”);
edit.commit();

SharedPreferences prefs = context.getSharedPreferences("details", MODE_PRIVATE);
String Uname = prefs.getString("username", "");
```

Reading/Writing Files

- Need to distiguish between files stored on internal storage (internal HDD) and externally (SD cards)
- For external storage WRITE_EXTERNAL_STORAGE is required

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

Internal storage	External storage
Always available.	Not always available as storage may be removed from the system.
Files saved here are accessible by the creating app by ***default***.	World-readable. Files saved here may be read/written/deleted outside of app.
When the app is uninstalled, the system removes all its files from internal storage.	When the app is uninstalled, the system removes files only if they were saved in the directory returned by getExternalFilesDir().
Internal storage is best suited for private data files.	External storage is best for files which are intended to be publicly accessible including accessible with a computer.

Reading/Writing Files

- *FileWriter f = **new** FileWriter("wrong.txt")*
  - not possible because of Android security system
  - throws *java.io.FileNotFoundException: /wrong.txt*
- Each APK file installed gets a User ID from the OS:
  - this ID is the key to the application's sandbox
  - the sandbox protects against (malicious) other apps
- Files created by an app are signed with its ID:
  - flags change access mode and make files accessible to other apps
  - Flags can be combined using the OR operator (|)

  MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE, MODE_PRIVATE, MODE_APPEND

File Writing - Internal Storage

- Write to a file with *Context.openFileOutput*:
  - file name (with possible path and optional mode)
  - returns a *FileOutputStream*

```
FileOutputStream fileOutput =
  openFileOutput("sample.txt", Context.MODE_PRIVATE);
OutputStreamWriter outputStream =
  new OutputStreamWriter(fileOutput);
outputStream.write("Hello Android");  ...
// ensure that everything is really written out
outputStream.flush();
outputStream.close();
fileOutput.close();
```


File Writing – External Storage

- Need to make sure external storage is available and writable: *storageState*
- *getExternalFilesDir() returns a location private to the app (however, READ_EXTERNAL_STORAGE permission allows to access all files on external storage, incl. data private to apps)*
- *Use getExternalStoragePublicDirectory() for public files*

```
String storageState = Environment.getExternalStorageState();
   if (storageState.equals(Environment.MEDIA_MOUNTED)) {
      File file = new File(getExternalFilesDir(null), “fileName");
      FileOutputStream fos = new FileOutputStream(file);
      OutputStreamWriter outputStream =
            new OutputStreamWriter(fos);
      outputStream.write("Hello Android");  ...
      // ensure that everything is really written out
      outputStream.flush();
      outputStream.close();
      fileOutput.close();
   }
```

File Reading – Internal Storage

- Reading a file with *Context.openFileInput*:
  - file name (with possible path and optional mode)
  - returns a *FileInputStream*
  - emulator files are stored in: */data/package/files*

```
FileInputStream fileInput = openFileInput("sample.txt");
InputStreamReader inputStream =
   new InputStreamReader(fileInput);
char[] inputBuffer = new char[128];
inputStream.read(inputBuffer);
String inputString = new String(inputBuffer);
inputStream.close();
fileInput.close();
```


File Reading – External Storage

- Again, need to check storage state
  - Does not need to be writable

```
String storageState = Environment.getExternalStorageState();
if (storageState.equals(Environment.MEDIA_MOUNTED) ||
    storageState.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
  File file = new File(getExternalFilesDir(null), “fileName");
  FileInputStream fileInput = new FileInputStream(file);
  InputStreamReader inputStream = new InputStreamReader(fileInput);
  char[] inputBuffer = new char[128];
  inputStream.read(inputBuffer);
  String inputString = new String(inputBuffer);
  inputStream.close();
  fileInput.close();
}
```

Databases

- Android uses the built-in SQLite:
  - open-source, stand-alone SQL database:
  - widely used by popular applications/systems
  - FireFox uses SQLite to store configuration data
  - iPhone uses SQLite for database storage
- A database is private to the application:
  - but data can be exposed through a content provider
- Emulator databases are stored in:
  /data/package/databases

Database Cursor

- The *Cursor* class is the return value for queries:
  - *Cursor* is a pointer to the result set from a query
  - this efficiently manage rows and columns
- A *ContentValues* object stores key-value pairs:
  - *put* inserts keys with values of different data types

Database Example

- Create a helper class to encapsulate particular aspects of accessing the database:
  - becomes transparent to the calling code
  - create, open, close and use the database
- Book database example:

id	isbn	title	publisher
0	158603524X	Using Android	Wiley
1	0201101947	Android Decoded	CRC Press

Database Helper

```
class DatabaseHelper extends SQLiteOpenHelper {

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


 @Override
 public void onCreate(SQLiteDatabase db) {
  db.execSQL(
    "create table " + DatabaseAdapter.DATABASE_TABLE  + "(" +
      "id integer primary key autoincrement," +
      "isbn text not null," +
      "title text not null," +
      "publisher text not null);"
  );
 }
 ... also overwrite onUpgrade(), and optionally onOpen()
```

Database Adapter

```
public class DatabaseAdapter {
  public final static String KEY_ISBN = "isbn";
  public final static String KEY_TITLE = “title";
  public final static String KEY_PUB = “publisher";
  public final static String DATABASE_NAME = "books";
  public final static String DATABASE_TABLE = "titles";
  public final static int DATABASE_VERSION = 1;

  private Context dbContext;
  private DatabaseHelper DBHelper;
  private SQLiteDatabase db;

  public DatabaseAdapter(Context context) {
    dbContext = context;
    DBHelper = new DatabaseHelper(context);
  }
  ...
```


Database Adapter: Inserting Data

```
  public DatabaseAdapter open() throws SQLException {
    db = DBHelper.getWritableDatabase();
    return(this);
  }

  public void close() {
    DBHelper.close();
  }

  // insert new book, returning new row identifier
  public long insertBook(String isbn, String title, String publisher) {
    ContentValues contentValues = new ContentValues();
    contentValues.put(KEY_ISBN, isbn);
    contentValues.put(KEY_TITLE, title);
    contentValues.put(KEY_PUB, publisher);
     return(db.insert(DATABASE_TABLE, null, contentValues));
  }
```

Database Adapter: Retrieving Data

```
// get a book, returning database pointer
public Cursor getBook(long rowId) throws SQLException {
  //parameters  to db.query include columns to return and condition
  //returned Cursor positioned before the first item, if not empty
  String where = “id=” + rowId;
  Cursor cursor = db.query(true, DATABASE_TABLE, where, ...);
return(cursor);
}
```


Database Adapter: Retrieving Data

SQLiteDatabase::query():

table	The table name to compile the query against.
columns	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
selectionArgs	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
groupBy	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
having	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
orderBy	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
limit	Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.

Database Adapter: Deleting/Updating Data

```
// delete a book , use WHERE argument with rowId
//returning true if it worked
public boolean deleteBook(long rowId) {
  String where = "id=" + rowId;
  return(db.delete(DATABASE_TABLE, where, ...) > 0);
}

// update a book, returning true if it worked
public boolean updateBook(long rowId, String isbn, String title,
 String publisher) {
  String where = "id=" + rowId;
  ContentValues contentValues = new ContentValues();
  contentValues.put(KEY_ISBN, isbn); ...
  return(db.update(DATABASE_TABLE, contentValues, where, ...) > 0);
}
```


Using The Database

- Instantiate *DatabaseAdapter* in the *Activity* constructor

```
public class MyActivity extends Activity {

 DatabaseAdapter db;

 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  db = new DatabaseAdapter(this);
  db.open();
  db.insertBook("1234567", "Brilliant Book", "Great Publisher");
  Cursor cursor = db.getBook(0);
  cursor.moveToFirst();
  while (!cursor.isAfterLast()) {
    String isbn = cursor.getString(0); String title = cursor.getString(1);
    cursor.moveToNext();
  }
}
```

Content Providers

- Files and databases are normally private:
  - unless specifically created otherwise
- Content providers provide data to other apps:
  - retrieve, modify and create data
- Example content providers in Android are *Contacts* and *MediaStore* (audio, images, video)
- Data is mapped to a URI used by clients:
  - *content://contacts/people/* all contact names
  - *content://contacts/people/23* contact with ID 23
  - *Uri.parse("content://contacts/people/23")*
  - *managedQuery(myPerson, ...)*

Customised Content Provider

- Extend *android.content.ContentProvider*
- Required methods need overriding:
  - *onCreate()* called when a provider is created
  - *getType(uri)* return MIME type of data
  - *insert(uri, contentValues)*
  - *query(uri, columns, selection, selectionArgs, sortOrder)*
  - *update(uri, contentValues, selection, selectionArgs)*
  - *delete(uri, selection, selectionArgs)*
- To create a CP for the earlier DB example, these methods can call the methods in the DatabaseAdapter class. → lab session