Android Programming
Lecture 16

11/4/2011
New Assignment

• Discuss New Assignment for CityApp
• GetGPS → Search Web Service → Parse → List Coordinates
Questions from last class

• It does not appear that SQLite, in the Android release, allows column-level encryption
  – Consider looking into SQLCipher
    [https://guardianproject.info/code/sqlcipher/](https://guardianproject.info/code/sqlcipher/)

• Size constraints
  – By default, SQLite limits to 2000 columns max in a table and 14 terabytes of data in a database
    • Clearly, phone device cannot hold 14 terabytes of data

• General encryption
  – Make use of javax.crypto classes (not specific to Android as part of general Java classes)
Creating a Database: Selecting Information

• Selecting information employs the “query” function of the database object

• Returns a Cursor, which is essentially an iterator over the returned values out of the table
  – Actually, it holds references to the values in the table
  – Use getXXX methods to extract data from Cursor
Creating a Database: Selecting Information: Query

• *query* function takes a lot of parameters: (book page 215)
  – A *boolean* indicating whether want only unique answers
  – String *name* of table to query
  – A *projection* (array of strings) that lists columns of interest
  – A *where* clause string that limits what data is returned (constrains based on value), supports wildcards (looks like: *attribute comparison value*)
  – An array of strings that can be used to list *wildcard substitutions*
  – A *group by string* clause that says whether certain rows should be returned together (usually used with an aggregate function (such as SUM))
  – A *having* string filter that constrains which groups to return (used with group by)
  – A string on which column and which ordering to use to *order* the data
  – A string optional *number of rows* limit

• Besides the first two, the others can be left as *null* and you will effectively retrieve the whole table queried.
• The *where* clause is how you strategically define what you want to get out
Creating a Database: Selecting Information: Cursors

- A Cursor has the ability to: (page 211 of textbook)
  - scroll back and forth over the returned values
  - ask about state of cursor in results
  - access data in the results

**Movement:**
- moveToFirst()
- moveToNext()
- moveToPrevious()
- moveToPosition()
- getPosition()

**Results meta-information:**
- getCount()
- getColumnNames()
- getColumnIndexOrThrow(String name)

**Data access:**
- getFloat(int index)
- getInt(int index)
- getString(int index)
- ...
Creating a Database: Selecting Information: Meals Example

• Retrieve all and print all but index data from Meals database:

```java
Cursor returnValues = mealsDatabase.query(false, "MEALS", null, null, null, null, null, null, null, null);
returnValues.moveToFirst();

do {
    String meal = returnValues.getString(returnValues.getColumnIndex("main")) + "," + returnValues.getString(returnValues.getColumnIndex("side")) + "," + returnValues.getString(returnValues.getColumnIndex("drink"));
    double price = returnValues.getDouble(returnValues.getColumnIndex("price"));
    outputView.setText(outputView.getText() + 
"\n" + "Paying " + price + " for " + meal);
} while (returnValues.moveToNext());
```
Creating a Database: Selecting Information: Meals Example

• Only print *main* components of meals over $4.00, sorted by descending price
  – Use a *where* clause to indicate only want rows where price > 4.00
  – Use an *order clause* indicating by price descending
  – Project onto just the *main* column

```java
String[] columnsOfInterest = new String[1];
columnsOfInterest[0] = "main";
returnValues = mealsDatabase.query(false, "MEALS", columnsOfInterest, "price > 4.00", null, null, null, "price DESC", null);
outputView.setText(outputView.getText() + "\n" + returnValues.getCount() + " rows returned.");
returnValues.moveToFirst();

while (returnValues.moveNext()) {
    String meal = returnValues.getString(returnValues.getColumnIndex("main"));
    outputView.setText(outputView.getText() + "\n" + meal);
} while (returnValues.moveToNext());
```
Other Database Commands

• Update row(s) in a table
  – Create ContentValue
  – Stuff with columns want to update
  – Use a *where clause* to indicate what row(s) to update
  – Call update function

  \[
  \text{databaseVariable.update(String tableName,}
  \text{ContentValue newValues, String whereClause, String[]}
  \text{whereClauseWildCardSubstitutions)}
  \]

// rename ChickenSandwich to McChicken
ContentValues toUpdate = new ContentValues();
toUpdate.put("main", "McChicken");
mealsDatabase.update("MEALS", toUpdate, "main='Chicken sandwich'",null);
Other Database Commands

• Delete rows in a table:
  – Provide a *where clause* indicating what to delete
  – `databaseVariable.delete(String tableName, String whereClause, String[] whereClauseWildCardSubstitutions)`

```java
// delete first entry
mealsDatabase.delete("MEALS", "id=1", null);
```
Big Picture

Most of the database functions are just abstracting out parts of the SQL language!
Saving Data On Phone

- Databases are one way of storing and accessing information on the phone
  - May actually be overkill for small or non-complex data
  - Alternatives exist:
    - SharedPreferences
    - Files in the Android Filesystem
SharedPreferences

• Common notions about apps:
  – Often allow users to set preferences for app
  – Often only a small number of settings
  – Should be respected across instantiations of app
    (including after app is killed, so can’t be stored in memory only)

  – For the Shuttle Tracking App,
    • Allow user to choose polling rate – how often to query
      where the bus is (5 seconds, 30 seconds, 1 minute, …)
      – Constrain them to choose from drop-down list
    • Remembers last bus viewed
SharedPreferences

• SharedPreferences store:
  – *You guessed it*: Key/value (label/value) pairs!
    • Keys are strings
    • Values are actual types
      – Supports: Boolean, String, float, long, integer

• SharedPreferences will be visible across all components of one app (across activities/classes within an app), and could be made visible to different apps (though not common)
SharedPreferences: Saving Data

- Retrieve the shared preferences
- Request to edit
- Store data using typed $\text{putXXX}(\text{label, value})$ methods
- Commit changes
SharedPreferences: Retrieving Data

- Retrieve the shared preferences
- Access using typed `getXXX(label,defaultValue)` methods
  - The `defaultValue` is returned if a value is not associated with a label in the preferences
  - `defaultValue` is important if user opening app for first time or user never changed (saved) preferences
SharedPreferences: Example

```java
package turkett.csc191;

import android.app.Activity;

public class RunCounterActivity extends Activity {

    TextView textView;
    SharedPreferences preferences;
    int runCount;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        textView = (TextView)findViewById(R.id.text_view);

        // print how many times the app has been restarted (fresh, after a kill) using shared preferences

        // retrieve preferences or default value
        preferences = getSharedPreferences("RUNCOUNTER", Activity.MODE_PRIVATE);
        runCount = preferences.getInt("runCount",0);
        Log.v("RC", "Read old preferences");
        textView.setText("This app has been run " + runCount + " times before.");

        // now since we know we have started (onCreate called on any start/ fresh restart) update count
        SharedPreferences.Editor preferenceEditor = preferences.edit();
        preferenceEditor.putInt("runCount", runCount + 1);
        preferenceEditor.commit();
        Log.v("RC", "Updated run count...");
    }
}
```
SharedPreferences vs. onCreate Bundle

How is this different from the Bundle that is passed to `onCreate()` and `onSaveInstanceState()` methods?

How could (should) we make use of SharedPreferences for the CityApp?
Direct File I/O

- Android API provides hook into the Android phone’s file system
- These methods to open files for input and output
  - Methods don’t allow folder specification
  - Typically employ app-specific permissions

- Can employ with general Java I/O classes and methods
Java I/O

- Java has two different styles of I/O
  - Streams – tend to be byte-based
  - Writers – tend to be text based

- Often wrap more complex streams/writers around simpler ones
  - Each acts as a bridge moving from high-level complex ideas (being able to print long strings) to low-level bits and byte manipulation
Android & Java I/O: Text Output

Common wrapping (using Android `openFileOutput`, which gives back a `FileOutputStream`)

```java
PrintWriter pw = new PrintWriter(
    new BufferedWriter(
        new OutputStreamWriter(
            openFileOutput(filenameGoesHere, Context.MODE_PRIVATE)));
```
Android & Java I/O: Text Output

`PrintWriter`: use `print()` and `println()` method to write arbitrary strings of text, will automatically call `toString()` on objects

`BufferedWriter`: handles buffering of data before writing to disk (writes in one big burst instead of each characters – important for efficiency)

`OutputStreamWriter`: handles String to byte conversion

`FileOutputStream`: writes bytes to a file on disk
Android & Java I/O: Example Text Output Code

```java
public class FileExamplesActivity extends Activity {

    TextView textView;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        textView = (TextView)findViewById(R.id.text_view);

        try {
            textView.setText("Writing to output file...");
            PrintWriter printWriter = new PrintWriter(
                new BufferedWriter(
                    new OutputStreamWriter(openFileOutput("testFile.txt", Context.MODE_PRIVATE)))
            );
            textView.setText(textView.getText() + "\nWriting "Hello World!"");
            printWriter.println("Hello World!");
            textView.setText(textView.getText() + "\nWriting "I wrote to a file on Android!"");
            printWriter.println("I wrote to a file on Android!");
            printWriter.flush();
            printWriter.close();
            textView.setText(textView.getText() + "\nWriting to output file complete.");
        } catch (Exception e) {
            Log.v("FIA", "File writing error...");
            Log.v("FIA", e.toString());
        }
    }
}
```

Java I/O code requires try/catch blocks

*Try:* Attempt to do file processing

*Catch:* If something goes wrong, let me know
Android & Java I/O

Files appear to be stored in this folder: `data/data/yourpackage/files/ filename`

Can use DDMS to move files on and off device