Accelerating Information Technology Innovation

http://aiti.mit.edu

Cali, Colombia
Summer 2012
Lesson 8 – Data binding and Databases

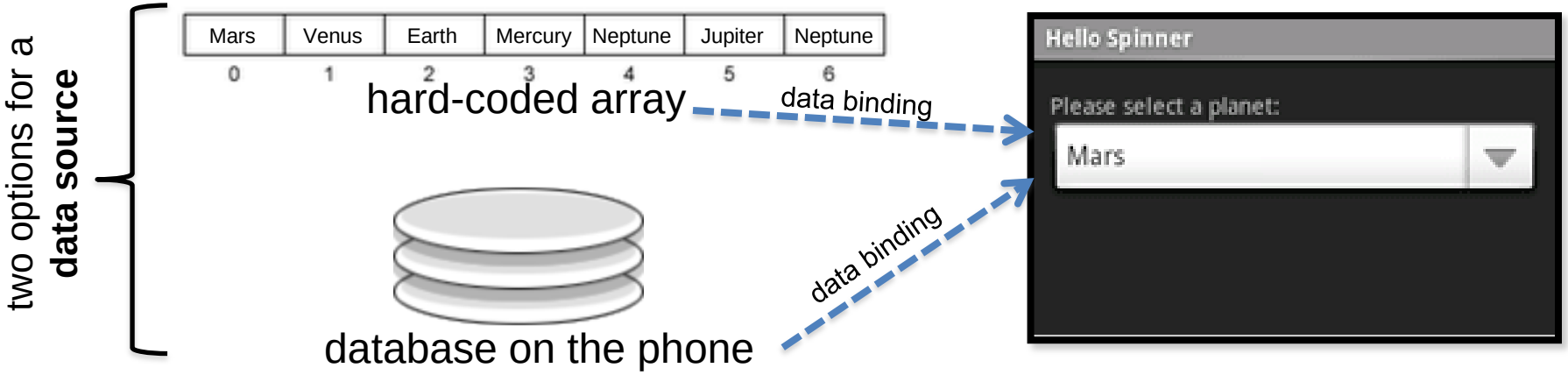
MISTI

Agenda

- Data Binding
- Databases in general
- Databases on Android (SQLite)
- A quick review of SQL
- Example of an Android app that uses databases

Importance of Data Binding

- **Data Binding**: the process of connecting views that display multiple items to a **data source**
- Any modifications to the data source will be reflected on the view immediately and automatically.



Possible Data Sources

- Data can be fetched from multiple sources:
  - Hard-coded arrays, defined in code
  - XML Resource Files
  - Databases on the phone
  - Content Providers / Content Resolvers (e.g. to populate a ListView with all the contacts on your phone)

Example: ListView with an array data source

- Step 1: Create a class of type ListActivitiy (as opposed to Activity)
- Step 2: Create the array data source. Two ways of doing this:
  - Hard-code the array in the ListActivity Class :

```
static final String[] THE_BIG_FIVE = new String[] {
    "Lion",
    "Leapord",
    "Rhino",
    "Elephant",
    "Buffalo"
};
```

  - Define the array in as an XML resource. Add the <string-array> to **`res/values/strings.xml`**

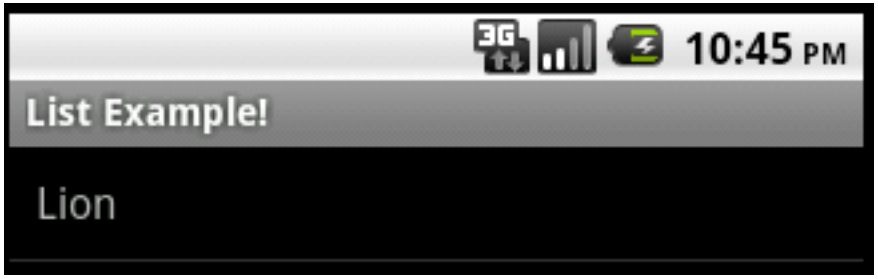
```
<resources>
    <string-array name="animals_array">
        <item>Lion</item>
        <item>Leapord</item>
        <item>Rhino</item>
        <item>Elephant</item>
        <item>Buffalo</item>
    </string-array>
</resources>
```

ListView Example, continued…

- Step 3: Create an XML layout file that will define how each cell or item in the ListView will look. Call this file “list_item.xml” and add to **`res/layout/`**

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

Note: This XML code means that each list item will essentially be a TextView, i.e. a simple text label. If we wanted each list item to also show an icon, we would need to modify this xml file to also include an ImageIcon and a layout of some sort.



Each list item, e.g. “Lion”, is simply a TextView

ListView Example, continued…

- Step 4: Now, establish the data binding in the onCreate() method of the ListActivity class
  - If data source is a hard-coded array, use the following:

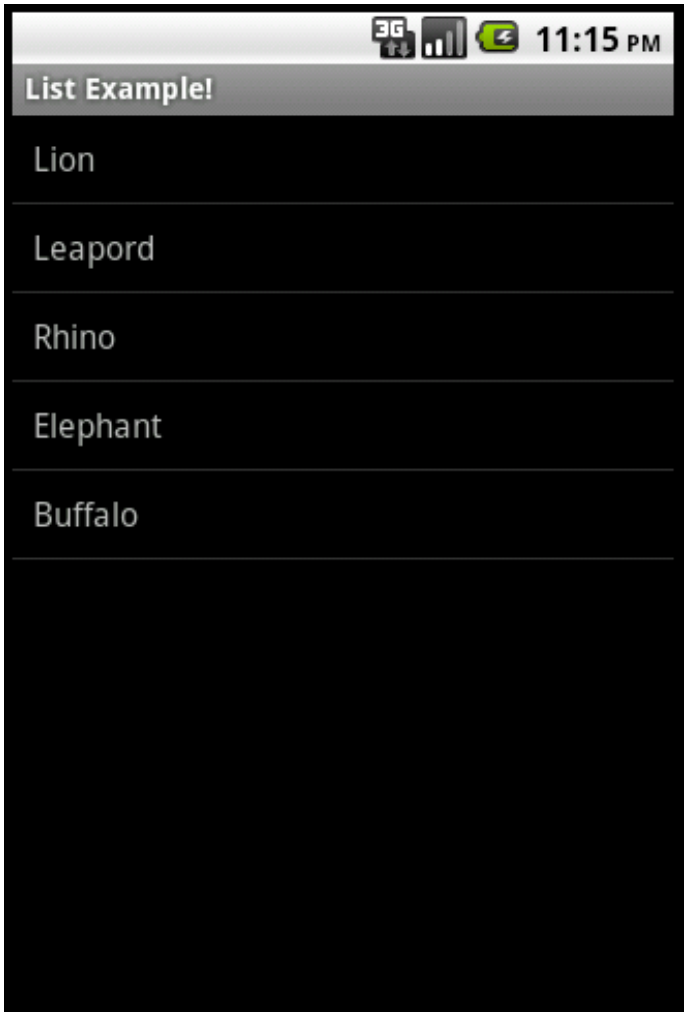
```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  //method 1
  setListAdapter(new ArrayAdapter<String>(this, R.layout.listitem_view, THE_BIG_FIVE));

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);

  lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
      // Handle list item click and do something here

    }
  });
}
```

ListView Example, continued…

- Step 4 contd…
  - However, If data source is defined in an XML resource file, use the following:

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  //method 2
  String[] students = getResources().getStringArray(R.array.students_array);
  setListAdapter(new ArrayAdapter<String>(this, R.layout.listitem_view, students));

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);

  lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
      // Handle list item click and do something here

    }
  });
}
```

The End Result!



Databases in general

- Database = data storage mechanism
- Useful for making data *persist* (keep track of data even when application is closed and reopened).
- Many different ways of implementing a database.
- One common approach: Relational Databases using **SQL** (a language used to insert, delete, and update data in a database)

Databases on Android (SQLite)

- The Android OS provides a built-in database management system called **SQLite** (a DB system specialized for embedded devices)
- Each Android application can have its own SQLite database, but may not access the database of any other application (for security)
- Advantages of SQLite:
  - Uses standard SQL syntax
  - Open-source, zero-configuration (no effort required by developer to set up the DB before using it)
  - SQLite system is not a client-server system (there's no SQLite server process that is always running).
  - Each SQLite database exists in its own, single file (very secure)

Quick review of SQL:

SQLite Database with two tables

Table 1: “notes”

_id	title	body
0	myFirstNote	Hi, abc…
1	anotherNote	blaablaablaa

SQL statement for creating table “notes”:

```
CREATE TABLE notes (_id integer primary key
autoincrement, title text not null, body text not
null);
```

Table 2: “employees”

_id	emp_name	emp_salary
0	Sally	$123,456
1	Bobby	$65,432

Quick review of SQL:

SQLite Database with two tables

Table 1: “notes”

_id	title	body
0	myFirstNote	Hi, abc…
1	anotherNote	blaablaablaa

SQL statement for inserting into tables:

```
INSERT INTO notes VALUES('myFirstNote', 'Hi,abc…');
INSERT INTO employees VALUES ('Sally', '123456');
```

Table 2: “employees”

_id	emp_name	emp_salary
0	Sally	$123,456
1	Bobby	$65,432

Quick review of SQL:

SQLite Database with two tables

Table 1: “notes”

_id	title	body
0	myFirstNote	Hi, abc…
1	anotherNote	blaablaablaa

SQL statement for selecting/deleting specific rows in the tables:

```
SELECT * FROM notes
   WHERE title = 'anotherNote'
   AND body = 'blaablaablaa';

DELETE FROM employees WHERE emp_salary < 100000;
```

Table 2: “employees”

_id	emp_name	emp_salary
0	Sally	$123,456
1	Bobby	$65,432

Example: Android Notes App

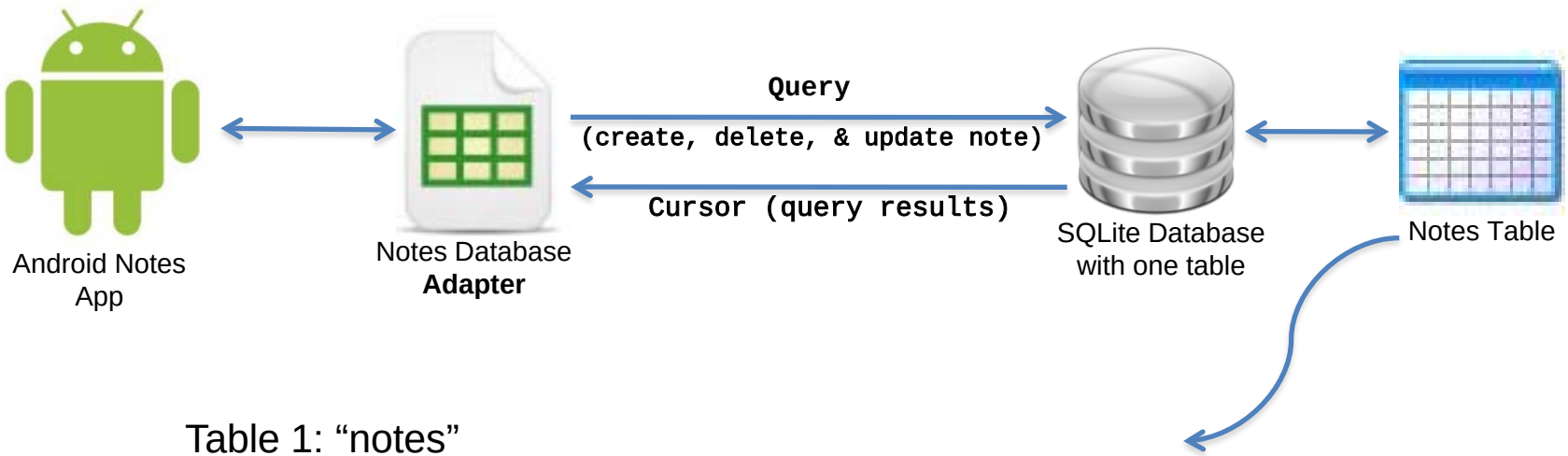


Table 1: “notes”

_id	title	body
0	myFirstNote	Hi, abc…
1	anotherNote	blaablaablaa

Example: Android Notes App

A closer look at the Notes Database Adapter

```
public class NotesDbAdapter {
    public static final String KEY_TITLE = "title";
    public static final String KEY_BODY = "body";
    public static final String KEY_ROWID = "_id";

    private static final String TAG = "NotesDbAdapter";
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;

    private static final String DATABASE_CREATE =
        "create table notes (_id integer primary key autoincrement, "
        + "title text not null, body text not null);";

    private static final String DATABASE_NAME = "data";
    private static final String DATABASE_TABLE = "notes";
    private static final int DATABASE_VERSION = 2;

    private final Context mCtx;

    private static class DatabaseHelper extends SQLiteOpenHelper { ... }

    public NotesDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }

    public NotesDbAdapter open() throws SQLException { ... }
    public void close() { ... }
    public long createNote(String title, String body) { ... }
    public boolean deleteNote(long rowId) { ... }
    public Cursor fetchAllNotes() { ... }
    public Cursor fetchNote(long rowId) throws SQLException { ... }
    public boolean updateNote(long rowId, String title, String body) { ... }
}
```

one constant for each column in the notes table

Example: Android Notes App

A closer look at the Notes Database Adapter

```
private static class DatabaseHelper extends SQLiteOpenHelper {

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

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS notes");
        onCreate(db);
    }
}
```

Example: Android Notes App

A closer look at the Notes Database Adapter

```
public class NotesDbAdapter {
    public static final String KEY_TITLE = "title";
    public static final String KEY_BODY = "body";
    public static final String KEY_ROWID = "_id";

    private static final String TAG = "NotesDbAdapter";
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;

    private static final String DATABASE_CREATE =
        "create table notes (_id integer primary key autoincrement, "
        + "title text not null, body text not null);";

    private static final String DATABASE_NAME = "data";
    private static final String DATABASE_TABLE = "notes";
    private static final int DATABASE_VERSION = 2;

    private final Context mCtx;

    private static class DatabaseHelper extends SQLiteOpenHelper { ... }

    public NotesDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }

    public NotesDbAdapter open() throws SQLException { ... }
    public void close() { ... }
    public long createNote(String title, String body) { ... }
    public boolean deleteNote(long rowId) { ... }
    public Cursor fetchAllNotes() { ... }
    public Cursor fetchNote(long rowId) throws SQLException { ... }
    public boolean updateNote(long rowId, String title, String body) { ... }
}
```

Example: Android Notes App

A closer look at the Notes Database Adapter

```
/**
 * Open the notes database. If it cannot be opened, try to create a new
 * instance of the database. If it cannot be created, throw an exception to
 * signal the failure
 *
 * @return this (self reference, allowing this to be chained in an
 *         initialization call)
 * @throws SQLException if the database could be neither opened or created
 */
public NotesDbAdapter open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();
    return this;
}

public void close() {
    mDbHelper.close();
}
```

Example: Android Notes App

A closer look at the Notes Database Adapter

```
public class NotesDbAdapter {
    public static final String KEY_TITLE = "title";
    public static final String KEY_BODY = "body";
    public static final String KEY_ROWID = "_id";

    private static final String TAG = "NotesDbAdapter";
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;

    private static final String DATABASE_CREATE =
        "create table notes (_id integer primary key autoincrement, "
        + "title text not null, body text not null);";

    private static final String DATABASE_NAME = "data";
    private static final String DATABASE_TABLE = "notes";
    private static final int DATABASE_VERSION = 2;

    private final Context mCtx;

    private static class DatabaseHelper extends SQLiteOpenHelper { ... }

    public NotesDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }

    public NotesDbAdapter open() throws SQLException { ... }
    public void close() { ... }
    public long createNote(String title, String body) { ... }
    public boolean deleteNote(long rowId) { ... }
    public Cursor fetchAllNotes() { ... }
    public Cursor fetchNote(long rowId) throws SQLException { ... }
    public boolean updateNote(long rowId, String title, String body) { ... }
}
```

Example: Android Notes App

A closer look at the Notes Database Adapter

```
/**
 * Create a new note using the title and body provided. If the note is
 * successfully created return the new rowId for that note, otherwise return
 * a -1 to indicate failure.
 *
 * @param title the title of the note
 * @param body the body of the note
 * @return rowId or -1 if failed
 */
public long createNote(String title, String body) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_TITLE, title);
    initialValues.put(KEY_BODY, body);

    return mDb.insert(DATABASE_TABLE, null, initialValues);
}

/**
 * Delete the note with the given rowId
 *
 * @param rowId id of note to delete
 * @return true if deleted, false otherwise
 */
public boolean deleteNote(long rowId) {

    return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}
```

Example: Android Notes App

A closer look at the Notes Database Adapter

```
public class NotesDbAdapter {
    public static final String KEY_TITLE = "title";
    public static final String KEY_BODY = "body";
    public static final String KEY_ROWID = "_id";

    private static final String TAG = "NotesDbAdapter";
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;

    private static final String DATABASE_CREATE =
        "create table notes (_id integer primary key autoincrement, "
        + "title text not null, body text not null);";

    private static final String DATABASE_NAME = "data";
    private static final String DATABASE_TABLE = "notes";
    private static final int DATABASE_VERSION = 2;

    private final Context mCtx;

    private static class DatabaseHelper extends SQLiteOpenHelper { ... }

    public NotesDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }

    public NotesDbAdapter open() throws SQLException { ... }
    public void close() { ... }
    public long createNote(String title, String body) { ... }
    public boolean deleteNote(long rowId) { ... }
    public Cursor fetchAllNotes() { ... }
    public Cursor fetchNote(long rowId) throws SQLException { ... }
    public boolean updateNote(long rowId, String title, String body) { ... }
}
```

Example: Android Notes App

A closer look at the Notes Database Adapter

```
/**
 * Return a Cursor over the list of all notes in the database
 */
public Cursor fetchAllNotes() {
    return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_TITLE,
            KEY_BODY}, null, null, null, null, null);
}

/**
 * Return a Cursor positioned at the note that matches the given rowId
 *
 * @throws SQLException if note could not be found/retrieved
 */
public Cursor fetchNote(long rowId) throws SQLException {
    Cursor mCursor = mDb.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
                        KEY_TITLE, KEY_BODY}, KEY_ROWID + "=" + rowId, null,
                        null, null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}
```

```
/**
 * The note to be updated is specified using the rowId, and it is altered
 * to use the title and body values passed in
 *
 * @return true if the note was successfully updated, false otherwise
 */
public boolean updateNote(long rowId, String title, String body) {
    ContentValues args = new ContentValues();
    args.put(KEY_TITLE, title);
    args.put(KEY_BODY, body);

    return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```

```
public class NotepadActivity extends ListActivity {

    private int mNoteNumber = 1;
    private NotesDbAdapter mDbHelper;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notepad_list);
        mDbHelper = new NotesDbAdapter(this);
        mDbHelper.open();
        fillData();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) { }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) { }

    private void createNote() {
        String noteName = "Note " + mNoteNumber++;
        mDbHelper.createNote(noteName, "");
        fillData();
    }

    private void fillData() {
        // Get all of the notes from the database and create the item list
        Cursor c = mDbHelper.fetchAllNotes();
        startManagingCursor(c);

        String[] from = new String[] { NotesDbAdapter.KEY_TITLE };
        int[] to = new int[] { R.id.text1 };

        // Now create an array adapter and set it to display using our row
        SimpleCursorAdapter notes =
            new SimpleCursorAdapter(this, R.layout.notes_row, c, from, to);
        setListAdapter(notes);
    }
}
```

Finally, change the Notepad App's Main Activity to interact with the database adapter we just created.

Note that the NotepadActivity is a ListActivity because the app displays the text of all saved notes in a ListView.