Mobile Software Development Framework: Android Activity, View/ViewGroup, External Resources, Listener

10/9/2012

Y. Richard Yang
Admin.

- Homework 2 questions
Recap: TinyOS

- **Hardware components motivated design**
  - Each component/module
    - specifies
      - the interfaces it provides
      - the interfaces/modules it uses
    - implements the functions in
      - the declared provided interfaces
      - event handlers in the declared used interfaces
  - Configuration specifies the linkage among components/modules

- **Execution model**
  - Event driven (triggered) handlers, who may post tasks to a FIFO task queue monitored by a task thread
Recap: J2ME Framework

- Java adaptation for mobile devices
- A major focus of J2ME is device heterogeneity
- Solution: versioning
  - Configurations
  - Profiles
Recap: J2ME MIDP Structure

Lifecycle callbacks
- startApp
- pauseApp
- destroyApp

```
d = Display.getDisplay(this)
d.setCurrent(disp)
```

MIDP

Current Displayable

A set of commands

Command listener

disp.addCommand()
Recap: IOS

- Apple’s adaptation for mobile devices
- E.g., Cocoa => Cocoa Touch
  - App KIT => UI Kit

Recap: iOS Model View Controller (MVC) Structure
Discussion

- What are some major differences between desktop GUI app design and mobile (phone/tablet) GUI app design?
Some Key Features of Mobile UI App

- Limited screen real estate
  - one thing at a time

- Limited resources: more dynamic system management on app life cycle
  - give app chances to adapt, better mem management

- More heterogeneous display devices
  - decoupling between display and logic
Mobile GUI App

- Screen real-estate is limited => Focus on one thing at a time
Mobile GUI App

- Screen real-estate is limited => Focus on one thing at a time
Mobile GUI App Workflow

App lifecycle callbacks/custom
- start
- pause
- ...

App
Mobile GUI App Workflow: Do One Thing

- App lifecycle callbacks/custom
  - start
  - pause
  - ...

App

Display Composite

Display

Display Composite

Display

Display

Display Composite

Display

Display
Mobile GUI App Workflow: Display Content Based on Underlining Data

App lifecycle callbacks/custom
- start
- pause
- ...

App

Display Composite

Display

Display Composite

Display

Display

Display

Data/Model

Data/Model
Mobile GUI App Workflow: Handle Events

App lifecycle callbacks/custom
- start
- pause
- ...

App

Display Composite
- Display
- Display Composite
- Display

Event Handler
- Event Handler

Data/Model
- Data/Model
Mobile GUI App Workflow: Switch to Another GUI

App lifecycle callbacks/custom
- start
- pause
- ...

App

Display Composite

Display

Display Composite

Display

Display Composite

Display

Display

Display

Display

Display

Event Handler

Event Handler

Data/Model

Data/Model
Discussion

- Key design points for mobile GUI app
  - Specify app life cycle customization
  - Specify display
  - Specify event scheduling
    - How to link event, display, handler, data
Typical Design: App

App lifecycle callbacks/custom:
- start
- pause
- ...

Framework reacts to app events and invokes app lifecycle event handlers.
How to Provide App LifeCycle Handlers?

- App class implements it
  - Inheritance

- App class does not implement it
  - Delegate
  - Command listener
Typical Design: UI

App lifecycle callbacks/custom
- start
- pause
- ...

System captures UI events; and puts them in a msg queue of the app. A UI thread processes the queue.
Example: IOS

Launch Time
- User taps app icon
- main()
- UIApplicationMain()
- Load main UI file
- Initialize the app

Foreground
- Activate the app
- Event Loop

Your code
- application: didFinishLaunchingWithOptions:
- applicationDidBecomeActive:
- Handle events
- Switch to a different app
How to Provide Display Component Event Handlers?

- Display Component class implements it
  - Inheritance
  - Typically a bad idea

- Display Component class does not implement it
  - Makes Display reusable
    - Delegate
    - Command listener
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
  - TinyOS
  - J2ME
  - IOS
  - Android
Android

- A mobile OS based on Linux
  - Customized Linux kernel 2.6 and 3.x (Android 4.0 onwards)
    - E.g., default no X Windows, not full set of GNU libs
  - Apply OS concepts for mobile contexts
    - e.g., each app is considered one Linux user
  - New key components, e.g.,
    - Binder for IPC
    - Power management wakelock

- Application development framework based on Java
  - Dalvik Virtual Machine
Android Architecture
Android Tools

- Android SDK Manager (android)
- Android emulator
- Android debug bridge (adb) can connect to an Android device and start a shell on the device

Mapping to Android

App lifecycle callbacks/custom
- start
- pause
- ...

- How to specify the customized callbacks: extend Activity class

- How to link the callbacks defined in view to listener/controller: View.set...Listener()

Allows external XML resource files to specify views
Application Framework (Android): Key Concepts

- Activity
- View/ViewGroup (Layout)
- External resources
**Activity**

- **A single, focused thing that the user can do.**
- **Creating a window to place UI views**
  - Full-screen windows, floating windows, embedded inside of another activity
- **Typically organized as a Stack**
  - Top Activity is visible
  - Other activities are stopped
  - Back button to traverse the Activity Stack
  - Long Home shows the content of the Stack
Activity: Manifest File

- To facilitate launching and managing Activities, each activity is announced in a manifest file.

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.hellotest">
    <application>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Instead of a hardcoded string in code, defines in res/strings.
Android Project Resources
// MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // savedInstanceState holds any data that may have been saved
        // for the activity before it got killed by the system (e.g.
        // to save memory) the last time

        super.onCreate(savedInstanceState);
        setContentView(…); // set a View

    }
}
A view component is a building block for user interface components.

Two types of views

- **Leaf**: TextView, EditText, Button, Form, TimePicker... ListView
- **Composite (ViewGroup)**: LinearLayout, RelativeLayout, ...
Programmatic Usage of Views

// MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // savedInstanceState holds any data that may have been saved
        // for the activity before it got killed by the system (e.g.
        // to save memory) the last time
        super.onCreate(savedInstanceState);

        TextView tv = new TextView(this);
        tv.setText("Hello!");
        setContentView(tv);
    }
}
Define View by XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</LinearLayout>
```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
}

TextView myTextView = (TextView)findViewById(R.id.myTextView);

main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/myTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello World, HelloWorld"
    />
</LinearLayout>
## External Resources

- Compiled to numbers and included in R.java file

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res/anim/</td>
<td>XML files for frame-by-frame animation</td>
</tr>
<tr>
<td>res/drawable/</td>
<td>images compiled and optimized</td>
</tr>
<tr>
<td>res/layout/</td>
<td>XML files for screen layouts</td>
</tr>
<tr>
<td>res/values/</td>
<td>compiled XML files into different resource</td>
</tr>
<tr>
<td>res/xml/</td>
<td>arbitrary XML files</td>
</tr>
<tr>
<td>res/raw/</td>
<td>raw, uncompiled files</td>
</tr>
</tbody>
</table>
Hello Example

- See HelloStart
Android Activity Life Cycle
Lifecycle Example

- See ActivityLifeCycle
Linking Views and Handlers/Controllers

- `onKeyDown`, `onKeyUp`
- `onTrackBallEvent`
- `onTouchEvent`

```java
myEditText.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (event.getAction() == KeyEvent.ACTION_DOWN)
            if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
                ...
                return true;
            }
        return false;
    }
});
```

```java
registerButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0) {
        ...
    }
});
```
Example: TipCalc

Set listener:

```java
public class TipCalcActivity extends Activity implements OnClickListener {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tip_calc);

        Button calc = (Button) findViewById(R.id.calculate);
        calc.setOnClickListener(this);
    }
    ...
}
```
Example: TipCalc

Handler:

```java
@Override
public void onClick(View arg0) {
    EditText amountText =
        (EditText)findViewById(R.id.amount_value);

    // get input
    double amt = Double.parseDouble(amountText.getText().toString());

    // compute output
    double tipD = amount * 0.15;

    // set UI
    String tipT = String.format("%.2f", tipD);
    TextView tipText = (TextView)findViewById(R.id.tip_value);
    tipText.setText(tipT);
}
```