

One Day of Mobile Application Development

Dino **Esposito**

Content

- Patterns/Practices of Mobile Development
- Programming for iPhone
 - Objective C, MonoTouch, PhoneGap
- Programming for Android
 - Java and SDK
- Programming for WP7
 - C# and XAML

Patterns of Mobile Development

(Mobile) Design Patterns

- Design patterns are general reusable solutions to recurring problems in software design
 - Description of a solution rather than working code
 - Triad: actors, algorithm, data
- Mobile design patterns
 - Desirable features to have in a mobile application
 - Platform independent (same for iOS, Android, and so forth)

Pattern Memento Mori (or app mortality)

Background

- Garbage collections applies to apps, not (just) objects
 - User only allowed to launch an application
 - System manages to quit the application
- One foreground application
 - System operates on app instances still active in the background
- Role and behavior of background apps may vary a bit
 - **iOS**: Just place calls to some specific API
 - **Android/BB**: Can buy extra processing time via a UI-less service. Not recommended for all applications (battery at risk)
 - **WP7**: Tombstoning and background agents

Pattern Memento Mori

Implementation

- Apps notified when no longer interactive
 - Given a few seconds of guaranteed lifetime
 - May be killed at any time
- Any relevant state at your fingertips
 - Present state and data to be likely used in the near future

Applications should consider saving their state when making it to the background

Pattern Back-and-Save

Background

- Youngsters quick like hell to type on mobile keyboards
 - ... but auto-completion is always welcome
 - ... and software that reads your mind is always welcome
- What's been just typed is anyway an effort the user made—don't throw it away because of Back or Search
- Minimize the typing effort of the user
- Use hints and smart forms of help to minimize errors
 - Should I say it? Validate input and don't trust users

Pattern Back-and-Save

Implementation

- Revert the classic pattern of desktop apps
 - Always save what's been typed
 - Offer a Clear button if that's important for the specific use-case
- Build your infrastructure to intercept when the user is leaving a screen—save and exit

Save the content of input forms when the user's leaving the screen

Pattern Cache-now-for-Later

Background

- Predictive fetch (or sliding download)
 - Try to intelligently guess what the user is going to do next
 - Try to download in advance data the user may need later
- *Later* is not necessarily in a few clicks
 - It can be hours or days until you get another chance to be online
- Example
 - Weather information
 - Don't limit to the current day; download the entire week
 - Apply "sliding download" policies

Pattern Cache-now-for-Later

Implementation

- Cache whatever can be used later
 - Data the user has typed
 - Choices made
 - State of the application
 - Fetch data in advance
- No guarantee of permanent connectivity

Remember data and activity and download data for later use ASAP & AFAP

Pattern Not-Now-Later

Background

- Data synchronization has always been a critical ingredient of mobile applications
- Occasionally connected applications are so common

Pattern Not-Now-Later

Implementation

- Manual coding using Web services
 - Take care yourself of deltas
- Sync Framework 4 for Windows
 - Now open sourced
 - Sync up with on-premise/cloud SQL Server
 - Not requiring clients on devices
 - Leverages a remote sync service and enables clients speaking HTTP and OData to use it
 - Usable on iPhone/Android as well

Pattern Guess-Don't-Ask

Background

- If there's something you can do to save users a click or typing, by all means do so
- Minimize interaction
 - Typing (use input scopes appropriately, auto-completion)
 - Tapping and clicking
 - Scrolling
 - Thinking (or make user's choices patently clear)
- Embed data that make the app start quickly
 - Define settings but provide reasonable default values for them

Pattern Guess-Don't-Ask

Implementation

- Remember preferences (cookie-like scenario)
- Example
 - Use geolocation to restrict searches
 - Arrange and use statistics about use of the app
 - Keep track of last action/selection
 - Link to contacts (if that's helpful)

Use any resources to make intelligent guesses and save users interaction

Pattern Login-and-Forget

Background

- Classic "**Remember Me**" scenario of Web applications
- Common pattern for applications that require a login to some Web service
 - Display a login box if credentials are not found on the device
 - If credentials are found, use stored credentials to log in automatically
 - Optionally, make your copy of credentials expire periodically (in addition to expiration policies set on the server)

Pattern Login-and-Forget

Security considerations

- The device can be lost or stolen
 - Another guy can pass himself off as you
 - Another guy can access the amount of info sitting in the phone (email, personal data, pics, contacts)
- Strong passwords are hard to type on mobile keyboards
 - Switch frequently between input scopes (digits, letters, symbols)
 - Subsequently, passwords are simpler than expected
- Credentials stored as clear text are not necessarily visible to anybody
 - Not on WP7; iOS has keychain repository
 - On Android, you should consider encryption/cipher

Pattern Login-and-Forget

Security considerations

- Behavior that simplify phishing is common in mobile
 - Click, click, and click
 - Blind clicking: don't read URL because of limited screen size
 - Harder to spot even patently suspicious URLs
- SSL for outbound communications not an issue on smartphones, but an issue on low-end devices
- Use platform-specific permissions
 - Principle of Least Privilege
 - Get just what you need; no more no less

Pattern 3-click Navigation

Background

- Immediacy is key in the mobile space
 - Users are not always comfortably sitting when they use the app
 - Walking, eating, driving, ...
 - Any action should be direct and quick
- Usability and design of the application
 - Well-defined use-cases
 - Detailed analysis of use-cases
 - Ask your kids about it; then make a second pass (as a dev)

Pattern 3-click Navigation

Implementation

- Split each screen in a few sections
- Make each feature ideally 3-clicks away
- Take this pattern as a vector, rather than a strict rule
 - But if fail on it, reconsider design and use-cases
- Ensure use-cases and user-stories match
 - Likely the user will find required steps «reasonable» and in a natural sequence
 - App and users on the same wavelength

Pattern The-App-Menu

Background

- Sometimes long list of items should be displayed
- Vertical lists of items are simplest approach
 - Works most of the time, because scrolling is an easy action on mobile devices
- Too long lists (100+ items) are boring to scroll
 - Create pages and scroll horizontally
 - You should stay focused on the OS standards

Pattern The-App-Menu

Implementation

- Imagine you're creating a restaurant menu
 - Create categories
 - List options and key information
 - Let users drill down

Pattern Babel-Tower

Background

- Which language(s) do you support?
 - Large audience == Large number of languages
- Many facets of localization
 - Text, Views, Graphics, Workflows
 - Text dwarfs everything else
- Native support is good but limited at "software" level
 - Bind strings to IDs and have some API to resolve them
 - Static approach—requires a new compile step
 - Make it more dynamic using some in-device database
 - Your API
- Main problem remains unresolved
 - How to get high-quality localized text?

Pattern Babel-Tower

Localization

- Not a new problem, but revamped by mobile applications
- Best-selling point of mobile apps is comfort for users
- Enabling users to play with the app in their own language is a double-edged sword
 - Great because users like it more
 - Bad, if translation is not appropriate
- Nearly all Web sites are limited to just a few languages
- Mobile apps are often offered in 10+ languages
 - Translating mobile apps is easier than a full-blown site

Pattern Babel-Tower

Implementation

- Download translated text on the fly
 - No need to update the app on the app store if you just add a new language to the list
 - Pick up the language based on the device settings ...
 - ... or your app will let users choose
- Enable (professional) translators to work on your text
 - Without conflict with development team and delays in the project management
 - Ship with the primary language and add new languages at your earliest convenience
- Keep an eye on **tiyla.com**

Check connectivity

- Never guaranteed—it comes and go quickly
- Use system notification services when available
- Refresh your UI promptly
- Always detect network availability and always have a plan B for network operations

iPhone Programming

iOS: tools

- You need a Mac; the cheapest Macbook is fine
 - Mac is necessary to compile the code as it relies on libraries that simply don't exist in Windows
 - Technically, can run OSX on a Win box; except that it is illegal 😊
- Join the iOS developer program (\$99/year)
 - Free registration doesn't not allow to test on real devices
 - In 2010, US declared jailbreaking lawful
- Get and install Xcode from Mac store
- Get and install the iOS SDK
- Make a decision about how to tackle iPhone programming ...

iOS: programming options

- Objective C
- Xamarin's MonoTouch for iOS
- Adobe's PhoneGap
- Appcelerator's Titanium
- Adobe's Creative Suite 5.5
 - Packager for iPhone and Android

\$399

\$699

\$49/month


iPhone App Basics

- Starter method (main.m)
- Single window object is created by you in *main* or loaded from a XIB file
 - XIB files are where the UI behind the iPhone application is saved/packaged (i.e., form designer files in .NET)
 - Single window contains views and/or controls
 - One window, possibly multiple views
- Views are rectangular areas on top a window
 - Display content (controls, animation, text, drawing)
 - Handle events (touch)
 - Various specialized views: table, Web, alert, navigation
 - View behavior handled by **controller** classes


```
#import <UIKit/UIKit.h>
#import "MyWindow.h"

int main(int argc, char **argv)
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    return UIApplicationMain(argc, argv, [Mywindow class], nil);
}
```

The window class to create.
This is **nil** if you rely on the main XIB file.



The name of the app-delegate to use.
If nil, then assumes "AppDelegate"



```
@interface Mywindow : UIApplication {
    UIView *mainView;
    UITextView *textView;
}
```

```
#import "Mywindow.h"
@implementation Mywindow

- (void) applicationDidFinishLaunching: (id) unused
{
    UIWindow *window;
    struct CGRect rect = [UIHardware fullScreenApplicationContentRect];
    rect.origin.x = rect.origin.y = 0.0f;

    window = [[UIWindow alloc] initWithContentRect: rect];
    mainView = [[UIView alloc] initWithFrame: rect];
    textView = [[UITextView alloc]
        initWithFrame: CGRectMake(0.0f, 0.0f, 320.0f, 480.0f)];
    [textView setEditable:YES];
    [textView setTextSize:14];

    [window makeKey: self];
    [window _setHidden: NO];
    [window setContentView: mainView];
    [mainView addSubview:textView];

    [textView setText:@"Hello world"];
}
```

Concepts you must get used to ...

- An **app-delegate** controls the behavior of the application from start to end
 - Receives notifications when the app reaches certain states such as “finished launching” or “willterminate” or “memory warning”
- A **view-controller** class governs the behavior of a view
 - Handles touch events, initialization

Concepts you may hear about ...

Outlets/Actions in Interface Builder

- Interface Builder is an optional tool that poor souls 😊 tend to match to Visual Studio designers
 - Make use of **outlets** to hide object references
 - Make use of **actions** to add event handlers
- Slightly tedious procedure
 - Control IDs are numbers
 - You create wrapper objects (outlets) and then must attach wrappers to specific controls
 - IB does that via drag-and-drop (details saved in XIB internals)
 - Alternative is creating the UI via code

Concepts you may hear about ...

Outlets/Actions

- An **outlet** is an “object reference” through which the controller acts with an object (i.e., button) in the view
 - Similar to Button1 members in VS, must be created explicitly
 - Need outlets to be able to set a label after a button click
- An **action** is an action to be performed on an object
 - First add outlets and actions to XIB
 - Next connect them to actual objects so that action “btnClicked” is associated with an event on Button1 and outlet “Button1” is associated with a given button...
 - Finally, write the code for btnClicked in the view controller class
- IB used for simple apps or simple views of an app
 - Can be told to autogenerate object-refs (i.e., .NET control IDs)

Concepts you may hear about ...

Outlets/Actions in code

- Real programmers do everything in code
 - Create control instances with explicit coordinates
 - Automatic z-order; layers go from background to foreground
 - Set properties (not all properties are visible through IB)
 - Set actions
- Everything takes place in *loadView* (for views)
 - In *applicationDidFinishLaunching* if you have just a window
- Real programmers often create a *just-for-fun.xib* to experiment with the graphical layout
 - Grab coordinates and copy to source code in *loadView*

```
(void) loadView
{
    // Create a full-screen view and set background
    UIView *view = [[UIView alloc]
                    initWithFrame:[UIScreen mainScreen] applicationFrame];
    view.backgroundColor = [UIColor whiteColor];

    // Create the button and set title and position
    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    button.frame = CGRectMake(100, 170, 100, 30);
    [button setTitle:@"Click"
               forState:UIControlStateNormal];

    // Add a click handler
    [button addTarget:self
              action:@selector(buttonPressed)
              forControlEvents:UIControlEventTouchUpInside];

    // Add the button to the view and set the view
    [self.view addSubview:button];
    self.view = view;
    [view release];
}

(void) buttonPressed
{
    NSLog(@"Button Pressed!");
}
```

iOS: MonoTouch

- Use .NET for building iOS applications
 - Check out Xamarin.com (\$399 license for individuals)
- A Mac computer is still required
 - Need: iPhone SDK, Mono, MonoTouch SDK
 - Use MonoDevelop to develop code
 - Use Interface Builder including Cocoa Touch thus having access to all the standard iPhone controls
- Limitations on generics and dynamic code (DLR)
 - No JIT in iOS
- You get a native iPhone application that can be published as any other iPhone application
- Wrappers for iPhone native API (accelerometer, GPS, ...)

iOS: MonoTouch

- Compile standard **.NET 4** code using the MonoTouch core assemblies
- Reuse is possible via a new compile step for the MonoTouch profile
 - Non UI-code
 - Code can potentially be shared between .NET, Android, and iPhone/iPad
- Currently, C# only
- With some work, it is possible to write iPhone code in Windows under Visual Studio and use the same project to compile on Mac

DEMO

- MonoTouch in action

iOS: Deployment

- Applications must be published to the AppStore
 - Internal distribution is possible only with an Enterprise developer account
- Application code must be signed against a distribution certificate (to identify the publisher) and a distribution provisioning profile
 - For companies, only the Team Agent can get the certificate
 - Get the AppStore distribution provisioning profile from the portal
 - Compile against that with your distribution certificate and submit

iOS: Testing on devices

- Get a Development Certificate to sign your code
 - Create a **Certificate Signing Request** using the Keychain Access application on your Mac
 - Log on to the portal and upload the CSR (**once per developer**)
 - Install the certificate in the Mac keychain
- Get a provisioning profile (Pprof)
 - Register a device manually through the portal or connect them to Xcode and let it do what's required (restricted to a few devices)
 - If you do it manually, you need the device UDID
 - Can get UDID via Xcode, iTunes, or the device itself (settings)
 - UDID != IMEI or serial number
- A Pprof can point to multiple devices (identified by UDID)

iOS: Getting the Pprof

- Xcode
 - Once the certificate is installed, you simply build the app and tell Xcode you want to test on the attached device
 - Xcode gets the Pprof automatically (if the device is registered)
 - In alternative, do it manually through the portal and download the Pprof to the device
- Ad hoc provision profiles
 - To test on non-registered devices (up to 100) create an ad-hoc provision profile manually on the portal
 - Indicate UDID and App ID and download the Pprof as a file
 - Compile the app against this Pprof and send both profile and app to the tester
 - Tester doesn't even need to have a Mac and install via iTunes

Over-the-Air Beta Testing

testflightapp.com

- Developer

- Get an account and create a team of testers (need their UDID)
- Create IPA against team members UDIDs and upload
- Go to site and distribute app
- Site makes your app available for download to members

- Tester

- Get an account and register device with the site (UDID needed)
- Get invited to test an app
- Receive via web a version of the app that runs on your device
- Outside marketplace

PhoneGap

- Open source solution for building mobile apps using HTML, JavaScript, CSS
 - No ASP.NET, Java, PHP; pure Web client solution
 - Create your app using HTML stuff (i.e., in VS)
- Shell of native code that hosts a set of HTML pages
 - Uses the WebBrowser control that each platform provides
 - Provides a JavaScript API to abstract device-specific functions
- No way to get automatically a native UI experience
 - Use CSS to mimic a native UI (e.g., iPhone)
 - Use jQueryMobile and touch libraries
 - Use script to build ad hoc controls (e.g., Android date-picker)

PhoneGap and JavaScript frameworks

- Device agnostic + open standards
- WebKit
 - Layout engine for rendering Web pages
 - Engine shared by nearly all mobile browsers (iOS, Android)
 - HTML5, local storage, CSS
- jQuery & jQuery Mobile
 - Better JavaScript programming
- XUI, Sencha Touch
 - Touch capabilities

PhoneGap

- MIT or BSD New license
 - Essentially, do whatever you want but place the "software provided as is" label in relevant parts of your source
- iOS, Android, BlackBerry (>4.5), webOS, Symbian
 - And growing
- PhoneGap Build
- Adobe

PhoneGap

- Native features reached via an internal bridge from PhoneGap JavaScript framework
 - Features depend on platforms/versions
- Geolocation
- Notification (sound, vibrate, alert)
- Storage
- Network
- Camera
- Accelerometer
- Contacts

PhoneGap for iOS

- Create a new PhoneGap-based application in Xcode
- Create a **www** folder in the Xcode project
- Edit the **index.html** which represents your main screen
- Add any Javascript, CSS and image files you need

- Build and run on simulator or device as usual

- PhoneGap apps are okay with Apple
 - Each app will be judged on its own merits, PhoneGap aside

DEMO

- PhoneGap in action

Android Programming

Android: Tools

- Pay a fee only to publish to the Market (one-time \$25)
- Get and install the Java SDK
- Get and install the Android SDK
- Eclipse or IntelliJ Community Edition as the IDE
- Get familiar with Java

Android App Basics

- Starter class
- Create main view
- View based on XML file(s)
- Event handling

- Manual binding of handlers to controls
- Manual definition of control references (outlets)

- Easy match with C#/VB

Android Programming

Activities

- Any app is based on one (or more) activity
 - Override method *onCreate* in any activity of yours
 - Perform any one-time tasks
 - Typically, register event handlers
- *onCreate* invoked each time the activity is created
 - When the application is starting up
 - When the application is re-launched after being paused
- App can be paused
 - Save/restore data is up to the application
 - Activity may be restarted when rotation changes

Android Programming

Layout and text

- UI of activities expressed in XML
- Relative positioning preferred
 - Absolute positioning deprecated

```
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="7dp"
        android:textStyle="bold"
        android:textSize="18px"
        android:textColor="#3ffffff"
        android:text="expoware.org/mobile welcomes you!" />
</LinearLayout>
```

Android Programming

Layout and text

- **LinearLayout**
 - Sequence of panels displayed horiz/vertically
- **RelativeLayout**
 - Elements indicate their preferred position relative to the parent
 - Right of, align-to-top, any number of child elements
- **FrameLayout**
 - Placeholder for a single (possibly nested) object, used to reserve space for dynamically generated content
- **TableLayout**
 - Row-based layout
 - Indicate rows, Android figures out ideal number of columns
 - Can indicate specific column a widget belongs to

Android Programming

Widgets and event handlers

- Widgets defined in XML or added programmatically
- Register listeners programmatically in onCreate

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

    // Instantiate internal members referring to widgets
    button1 = (Button) findViewById(R.id.button1);

    // Attach event handlers as required
    button1.setOnClickListener(button1Click);
}

private View.OnClickListener button1Click = new View.OnClickListener() {
    public void onClick(View v) {
        updateScreen();
    }
};
```


DEMO

- Android in action

Android Programming

Menus

- Define menus as XML resources
 - ID, text and icon
 - Create submenus by nesting <menu> sub-trees
- Group logically related menu items
 - Disable and enable items together
 - Use <group> element to group multiple menu items
 - Each group is given a unique ID
 - No impact on rendered UI; purely logical
- Menu overrides
 - onCreateMenuOptions and onPrepareMenuOptions
 - Inflate menu to display
 - onOptionsItemSelected

Android Programming

Dialogs

- Auto-disappearing toast messages for quick feedback
- Alert dialog boxes: boring to deal with ...

```
AlertDialog.Builder builder = new AlertDialog.Builder(MyActivity.this);
builder.setMessage(message)
    .setCancelable(false)
    .setTitle(title)
    .setPositiveButton(yes, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            StartNewGame();
            dialog.dismiss();
        }
    })
    .setNegativeButton(no, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
AlertDialog alert = builder.create();
alert.show();
```

Android Programming

Storage: preferences

- Preference framework
 - A preference is value+key+description+default value
 - Use XML to define groups of preferences
 - Types of preferences as UI tips (checkbox-pref, edittext-pref, list)
- Load preferences into an activity
 - Both main activity and additional activity
 - Activity displays an ad hoc UI built around preferences
- Automatic edit and save
- Data saved to a local file—transparent to users/devs

Android Programming

Storage: preferences

- Preference framework

```
// Trigger a new activity with the preference screen
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    Intent intent = new Intent(this, LoginPreferences.class);
    startActivity(intent);
}
```

```
public class LoginPreferences extends PreferenceActivity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.NameOfTheXmlFile);
    }
}
```

Android Programming

Storage: preferences

- Direct access to the preference API

```
// Save preferences (decide about file name and visibility)
SharedPreferences preferences;
preferences = getSharedPreferences(fileName, MODE_PRIVATE);
Editor preferenceEditor = preferences.edit();
preferenceEditor.putString(key, text);
preferenceEditor.commit();
```

```
// Read preferences (use default storage)
SharedPreferences preferences;
preferences = PreferenceManager.getDefaultSharedPreferences(context);
info.Nickname = preferences.getString("nickname", null);
info.Password = preferences.getString("password", null);
info.RememberMe = preferences.getBoolean("rememberme", true);
```

DEMO

- Android in action

Android Programming

Storage: files

- Data saved to files is considered private of the app
 - Removed when you uninstall/clear the app
 - A file is a file—public, unless you mark it as private
 - Files and directories rooted in `"/data/data/APP/files"`
- Access to external storage (SD) requires permission
 - Slightly different API for SD cards
 - SD may not be available during debug
- Classic stream-based API
- Learn and apply object serialization

Android Programming

Storage: serialization

```
public class ObjectFormatter {
    public static byte[] Serialize(Object o)
    {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        try {
            ObjectOutputStream out = new ObjectOutputStream(bos);
            out.writeObject(o);
            out.close();

            // Get the bytes of the serialized object
            byte[] buf = bos.toByteArray();
            return buf;
        } catch(IOException ioe)
        {
            return null;
        }
    }
}
:
```

Android Programming

Storage: deserialization

```
public class ObjectFormatter {
    public static Object Deserialize(byte[] b)
    {
        try {
            ObjectInputStream in = new ObjectInputStream(new
                ByteArrayInputStream(b));
            Object object = in.readObject();
            in.close();
            return object;
        }
        catch(ClassNotFoundException cnfe) {
            return null;
        }
        catch(IOException ioe) {
            return null;
        }
    }
}
```

Android Programming

Storage: picklist

- Alert dialog with a list of items as argument

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a match");
builder.setItems(matches, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int index) {
        String fileName = matches[index];
        MatchStatus status = StoreManager.ReadFromPrivateFile(
            getApplicationContext(), file);

        // update the UI of the application to reflect the selected match
        SetMatchStatus(status);
        RefreshView();
    }
});

AlertDialog alert = builder.create();
alert.show();
```

DEMO

- Android in action

Android Programming

Networking

- Add permission "android.permission.INTERNET"
- Clean HTTP API

```
url = "...";
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(url);
HttpResponse response = client.execute(request);
BufferedReader rd = new BufferedReader(
    new InputStreamReader(response.getEntity().getContent()));
String line = "";
while ((line = rd.readLine()) != null) {
    DoSomethingWithLine(line);
}
```

```
HttpPost request = new HttpPost(url);
request.setEntity(new UrlEncodedFormEntity(nameValuePairs));
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
nameValuePairs.add(new BasicNameValuePair("id", "12345"));
nameValuePairs.add(new BasicNameValuePair("data", "Android is cool!"));
HttpResponse response = httpClient.execute(request);
```

Android Programming

Network availability

- Key aspect of mobile programming

```
// If no network is available networkInfo will be null,  
// otherwise check if we are connected  
public boolean isNetworkAvailable()  
{  
    ConnectivityManager cm;  
    cm = (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);  
  
    NetworkInfo networkInfo = cm.getActiveNetworkInfo();  
    if (networkInfo != null && networkInfo.isConnected())  
    {  
        return true;  
    }  
    return false;  
}
```

Android Programming

SMS

- Add permission "android.permission.SEND_SMS"

```
// send SMS silently
SmsManager sm = SmsManager.getDefault();
String number = "...";
sm.sendTextMessage(number, null, "Your message", null, null);

// Show the screen for sending SMS and let user type.
// No need to get permissions for this
String number = "...";
startActivity(
    new Intent(Intent.ACTION_VIEW, Uri.fromParts("sms", number, null)));
```

Android Deployment

- Just compile and distribute the APK executable
- Everything is at your own risk
 - Behavior over different firmware
 - Different hardware/software characteristics
 - Conflicts with other applications and crashes
- Android requirements
 - Upload signed code (when publish the app)
 - Sign against a (self-issued) certificate created with SDK tools
 - Re-sign with the same certificate when updating an app
 - Not necessary to sign all of your apps with the same key
- My perspective of the Android jungle

Windows Phone 7 Programming

WP7 Programming

Startup

- App.xaml and WMAppManifest.xml
- **PhoneApplicationFrame** is the top level container
 - One for the entire application
 - Created automatically when the application is initialized
 - Load the page specified in WMAppManifest.xml
- **RootVisual** property to set the main application UI
 - Dismisses the splash screen
- PhoneApplicationPage is the base class for pages

WP7 Programming

Layout and XAML

- Application bar
- StackPanel, Grid, Canvas
- Style
- Controls and Templates
- User Controls
- Storyboard
- Binding syntax

WP7 Programming

Pivot and panorama

- Pivot applications
 - Tab like applications
 - Collection of pages loaded individually

- Panorama applications
 - Single page with multiple panes
 - Horizontal scrolling
 - Memory intensive: unique multi-pane page
 - Never more than 3-4 panes

DEMO

- WP7 in action
 - (Snippets of a real-world app)

WP7 Programming

Isolated storage

- Same API as in .NET
 - Stream-based
 - Serialization of objects
- Inaccessible to other applications
- Need to use a special service to explore storage

- SQL Server CE writing files within iso-storage
 - Windows Phone 7.5 (Mango)

- Best approach: Tell-Don't-Save-Over-and-Over-Again
 - Build a framework to save your classes and reuse the code

WP7 Programming

Back and save

- OnBackPressed on pages
 - Allow to cancel the back navigation
 - Can be used to save anyway data
- Back and pivot/panorama
 - Move back to the first item instead of exiting the app
 - Dismiss dialog boxes (up to you to create a framework)
- Save state during navigation

WP7 Programming

Launchers and choosers

- Launch a system dialog
 - WebBrowser, Email, MediaPlayer, PhoneCall, Sms, Search
- Launch a system dialog and get a value
 - CameraCapture, EmailAddress, PhoneNumber, Photo

DEMO

- WP7 in action
 - (Snippets of a real-world app)

WP7 Programming

Networking

- Async model
- WebClient and HttpWebRequest
- JSON serializer
 - DataContractJsonSerializer

Hard termination of an app

Exit via the Back button

- Quit the application and free memory
- Apps get notifications for the event
 - Chance to save state to permanent storage
 - App decides what's relevant
- Restore last known state and provide a continuous-feel experience

Soft termination of an app

User activity causes termination

- Incoming calls
- Engaged screen
- Tap **Start** button (i.e, launch another app)
- Tap **Search** button
- Respond to toast notifications
- Programmatically invoke launcher/chooser

Tombstoning

- Apps given a 10 sec notice of termination
 - Save state to transient memory
 - Save state to permanent storage
- Apps are then removed from memory
 - System retains app's transient memory
 - Data stored as long as possible

Resuming from tombstoning

Scenarios

- User ends the phone call
- User disengages the screen
- User navigates back to the application using the Back button
 - After checking a toast message
 - After making a search
- User completes given chooser task

Resuming from tombstoning

Action

- User reactivates a tombstoned app
 - New application instance created
 - Transient state (if any still available) passed
 - Regular initialization process bypassed
- No guarantee transient state is still there
 - If not, regular initialization takes place

Behavior of tombstoned apps

- No behavior at all; just dead code
 - Only MS native apps allowed to be active in the background
- Fake multitasking allows users to switch apps in a few seconds—**not instantly**
- Change in WP7 Mango

PhoneApplicationService

Events

- **Launching**
 - Application being launched
- **Closing**
 - Application is exiting
- **Deactivating**
 - Being tombstoned
- **Activated**
 - Made active after being tombstoned

Lifecycle

Standard launch/close cycle

- Launching
 - Do not load content from storage or Web
 - Do not assume the app is resuming from a previous session (check StartupMode)
- Closing
 - Save persistent state, if any

Lifecycle

Tombstoning cycle

- Deactivated
 - Save transient state into the State dictionary
 - Consider saving application state to storage
 - 10 seconds to complete
- Activated
 - Loading any transient state
 - Avoid retrieving data from storage
 - Critical for load time (consider optimization)

MVVM

- MVVM and Blend
- Just a matter of data-binding?

General practices

Layout

- Based on use-cases, define the main screen for the app
 - Use the typical layout of the platform
 - Just a few buttons for the main options
 - Use menu if common on the platform
 - Ensure you can control **easily** the layout
- Add pages and navigation
 - Handle Back button properly
 - Use popup dialogs when it's a quick selection
- Add splashscreen and about

Settings

- Abstract your app settings in a class
 - Add serialization logic to the class
- Use settings throughout the app
 - Bind to localization logic if required
- Add a Settings page
 - Save state and settings as often as possible

Design consideration

- Mobile apps are relatively simple
- Not a good reason to release on design constraints
- Maintainability
 - Expect frequent updates and extensions to the code

Summary

- *Mobile development has a lot of common ground between platforms*
- *Each platform does things in a different way—but nearly the same things*
- *IDE is problematic more than languages and SDKs*