600.250 USER INTERFACES AND MOBILE APPLICATIONS

Joanne Selinski

Spring 2011

Course Topics

- Teamwork Skills
- User Interface Design
- Mobile Application Development
- Testing & Debugging
- Social Impact

Teamwork

GOOD

- communication – lots
- diversity – play to different strengths
- compromise

BAD

- communication – not enough
- uniformity – missing needs
- lack of version control
- control freaks, egos
- lack of confidence

User Interfaces

GOOD

- intuitive
- clean
- simple
- elegant
- right level of information
- fast

BAD

- overcrowding
- too complicated
- poor aesthetics
- lack of response, feedback
- pop-ups
- physically unwieldy

HCI, UX, UI, oh my!

- HCI: human computer interaction
- UI: User Interface
- GUI: Graphical User Interface
- UX: User Experience

What is Android?

- Operating System
- lots of phone platforms
- and tablets
- google

Why Android?

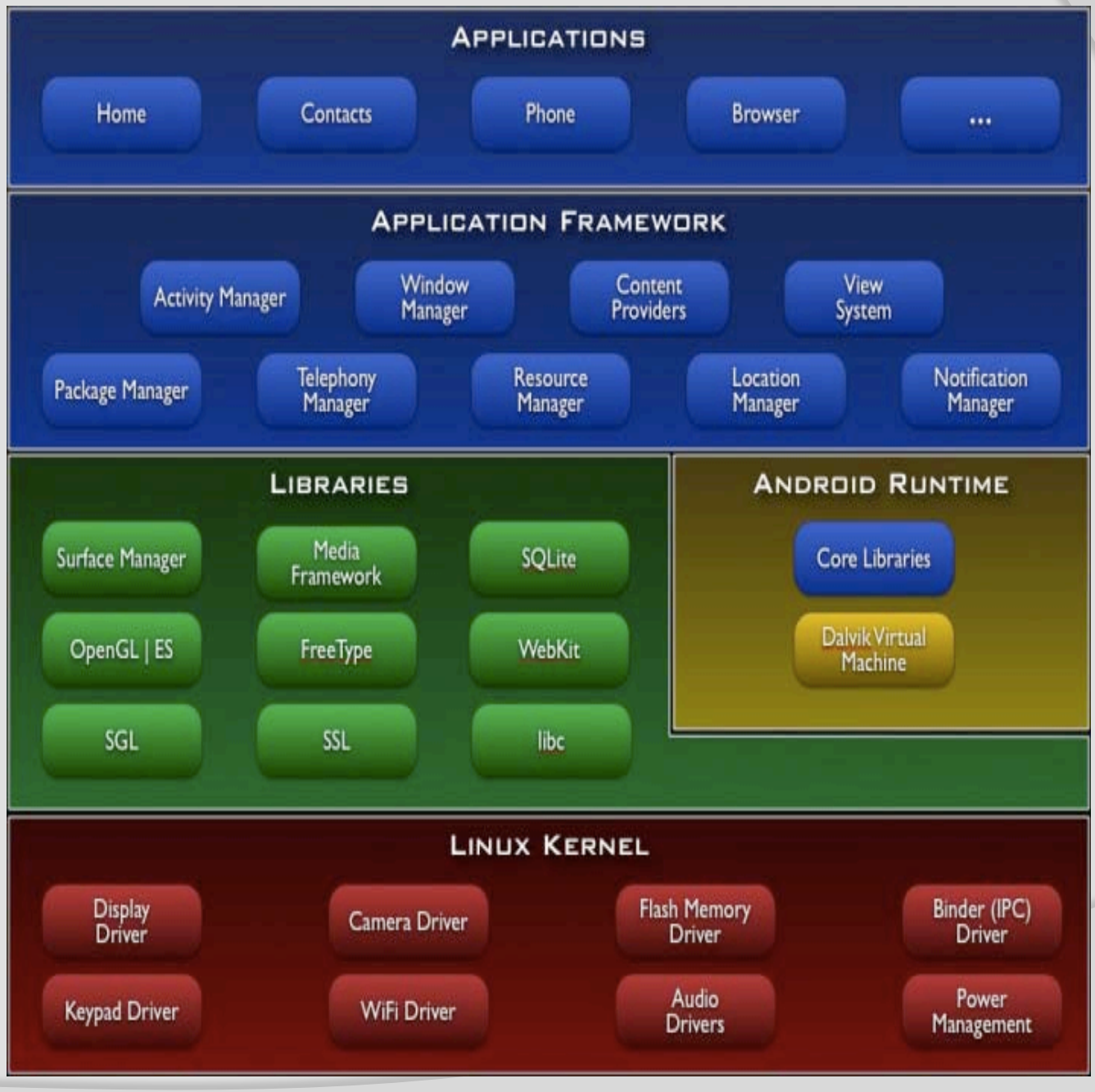
- open source
- lots of samples
- popular
- free to develop
- java based
- not a walled garden
- easily shared

How Android?

- Eclipse
- Emulator
- Code Samples

Android Software Stack

- key applications
  - user applications
  - application framework
- middleware
  - libraries
- operating system
  - linux kernel
- android runtime


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Android Applications

- devices ship with core apps, eg:
  - home
  - contacts
  - browser
  - texting (SMS)
- we download and write our own, eg:
  - HelloWorld
  - AngryBirds
  - Sudoku

Application Framework

- we can write apps that access any phone features, same as the core apps
- designed for component reuse
- apps can make capabilities available to other apps
- components can be replaced (customized)

App Services & Systems

- Views for UIs
  - primary application components
  - lists, grids, text boxes, buttons, etc.
- Resource Manager provides
  - localized strings, graphics, layouts
- Content providers enable
  - stored data sharing (eg, Contacts)
- Notification Manager provides
  - consistent, non-intrusive signaling
- Activity Manager controls
  - application lifecycles
  - navigation backstack

Android Runtime

- set of core libraries
- Dalvik virtual machine (DVM)
  - an instance for each application
  - register-based
  - optimized for concurrently running instances
  - executes *.dex (Dalvik executable) format
  - dx tool compiles Java into *.dex
  - relies on the linux kernel
  - (advanced developers can use C/C++ and OpenGL directly)

Android Development Features

- Application framework
- Dalvik virtual machine optimized for mobile devices
- Integrated browser
- SQLite for structured data storage
- Rich development environment (Android SDK, Eclipse plug-in, emulator)

Android Hardware Features*

- Optmized graphics
- Media support for audio, video, and still images
- GSM Telephony (Global System for Mobile Communications)
- 3G
- WiFi
- Bluetooth
- EDGE (Enhanced Data GSM Environment)
- Camera
- GPS
- Compass
- Accelerometer

*not all phones have all features

Android Application Types

- foreground applications
  - need cool UIs (sudoku)
- background services & intent receivers
  - little to no user input (back-up assistant)
- intermittent applications
  - combination of visible & invisible (music)
- widgets
  - dynamic data displays (date/time)
- complex apps may be combinations

Android Application Components

- activities
  - windowed visual UIs
  - built with hierarchy of views
- services
  - run in background, no visual UI
- content providers
  - shareable data stores
- broadcast receivers
- notifications
- widgets

Android Intents

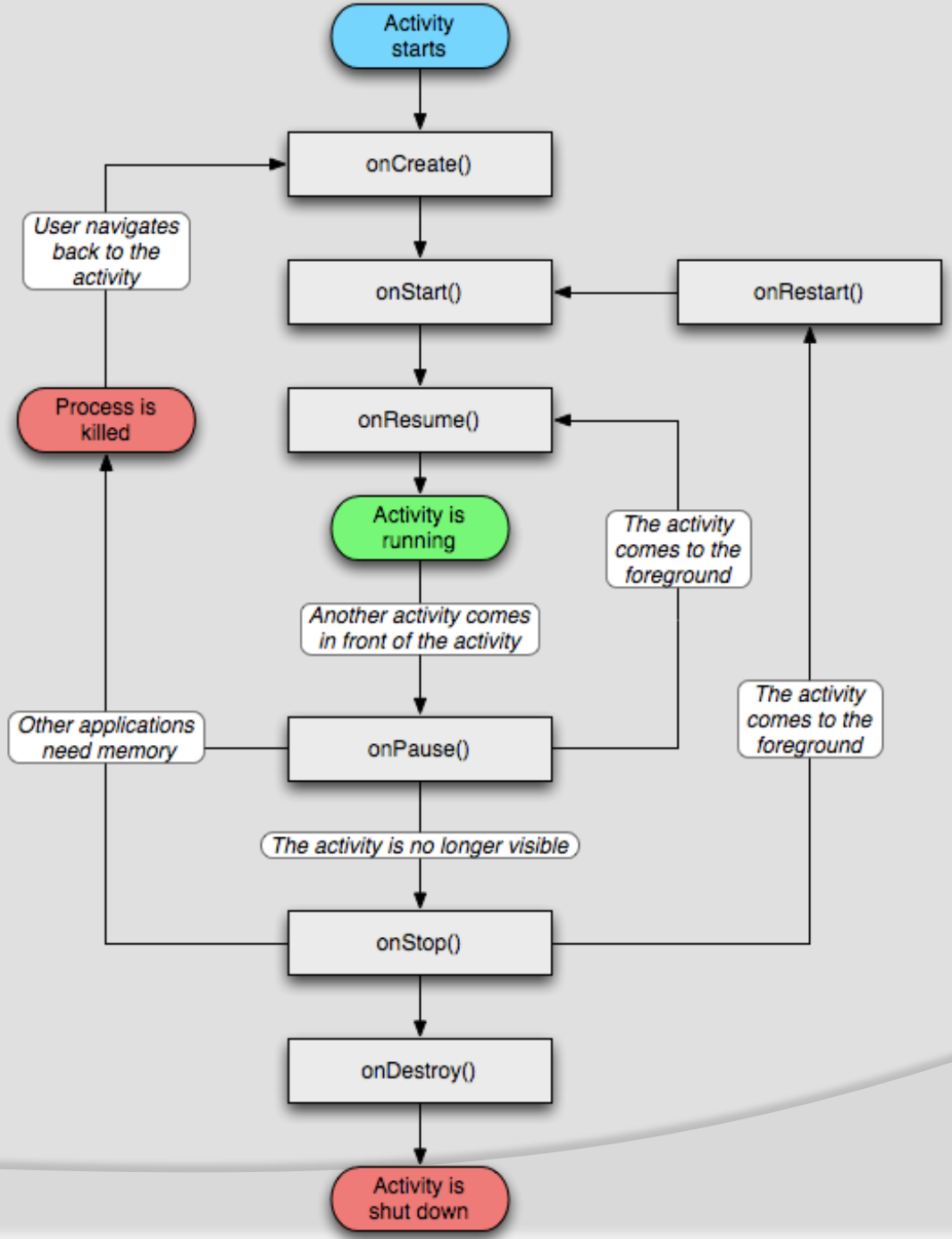
- asynchronous messages
- used to make things happen
- activate 3 component types:
  - activities
  - services
  - broadcast receivers

Phone Demo

- foreground app: calendar
  - activities: day, week, month views, create event, choose time, reminders, repeat
  - notification: reminder alerts
- foreground app: contacts
  - activities: scroll contacts, edit contact, create
  - content provider: used by texting, phone, email
- background app: back-up assistant
  - service: sends stored contact updates to Verizon
  - notification: if auto-update fails
  - activities: set schedule
- intermittent app: music
  - activities: select, start/stop
  - service: plays when not in foreground
  - broadcast receiver: interrupted if call comes in
- widgets: weather

Activity States

- An activity has essentially four states:
  - Active or Running: in the foreground
  - Paused: partially obscured, system kill in extreme memory situations
  - Stopped: completely obscured, often killed by the system when memory is needed
  - Killed: the system can drop a paused or stopped activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Activity Lifecycle

```
public class Activity extends ApplicationContext {
    // full lifetime
    protected void onCreate(Bundle savedInstanceState);
        // visible lifetime
        protected void onStart();
        protected void onRestart();
            // active lifetime
            protected void onResume();
            protected void onPause();
        protected void onStop();
    protected void onDestroy();
}
```

References

- http://developer.android.com
- Reto Meier, *Professional Android 2 Application Development*, Wiley, 2010

Fundamental Coding Concepts

- interface vs. processing
  - separation is key!!!
  - android app interface: res files
    - layout *.xml to define views
    - values *.xml to define strings, menus
    - drawable *.png to define images
  - android app processing: src *.java
- event based programming
  - javascript example?

Android "Under the Hood"

- snake sample code
  - relatively simple animation
- notepad sample code
  - multiple menu system
  - content provider

Open Source

- many licenses
- not always required to maintain openness
- must reiterate license agreement in all distributions
- Android primarily uses Apache 2.0: http://source.android.com/source/licenses.html

Free Software

- GNU license
- eg: linux, gcc
- freedom
  - to reuse
  - to distribute
- license – share under same conditions (viral)
- "copyleft"

Application Manifest

- master .xml file
- declares all the components of the application
- declares intent filters for each component
- declares external app info: name, icon, version, SDK levels, permissions, required configurations and device features, etc.
- http://developer.android.com/guide/topics/manifest/manifest-intro.html

Creating a New Application I

Create Project

- File -> New -> Project -> Android -> Android Project
- New Project Wizard to specify
  - project name – file folder
  - check/change location
  - choose lowest level build target
  - application name – friendly description
  - package name – 2 part java namespace
  - create activity – main activity component
  - min SDK – lowest level that app will run on

Creating a New Application II

Create Launch Configuration

- project and activity to launch
- virtual device and emulator options
- input/output settings

Creating a New Application III

Run, Edit, Debug

- debug & run default Hello World activity
  - cmd-shift-O to import necessary classes in Eclipse
  - select project, right click, Android Tools -> Fix Project Properties
- edit the starting activity
- implement new components and behaviors (borrow from samples)
- select the project and then

  Run -> Run As -> Android Application

Code Writing Approaches

- start with exact plan of what components and elements to use, find a way to make it happen from scratch -> ok for experts

- start with a rough idea of what features and elements you want, find samples that are close or similar, adapt to fit -> better for beginners

Teamwork (revisited)

GOOD

- communication – lots
- diversity – play to different strengths
- compromise

BAD

- communication – not enough
- uniformity – missing needs
- lack of version control
- control freaks, egos
- lack of confidence

Teamwork - Communication

GOOD

constructive criticism

"could be better if"

offer alternatives

take burden on yourself
"not working for me"

restate for clarity

give approval

"i'm not a big fan of..."

BAD

"sucks"

shooting something down

silence, lack of response

assume other know what you're thinking

bite your tongue

"i hate this"

Individual Responsibilities

- stay on track – meet deadlines
- stay on track – don't digress
- stay on track – stick with requirements
- thorough documentation
- save code before major changes
- consult teammates before changes
- make time for meetings
- come to meeting prepared
- be on time
- compromise
  - meet in middle
  - take turns
  - agree to disagree
- be respectful
- quality is important
- do your share of the work
- communicate!
- give and take – help
- distinguish between important and trivial

Teamwork Process Essentials

- realistic requirements
- decomposition
- deadlines
- integration

(from Effective UI)

Teamwork – Coding Techniques

- Paired programming
  - driver
  - navigator
- Ruthless testing
  - unit tests
  - android test features

Teamwork - Sharing

- Documents
  - jShare
  - code versioning (to be visited later)
  - dropbox
  - google docs
- Ideas
  - discussion list/email group alias
  - google wave?
  - face to face – brainstorming session

Teamwork - Diversity

- technical skills
  - programming (coding) experience
  - presentations
  - testing, debugging
  - designing
- personalities
  - leadership
- backgrounds?
- friendship – too much bad, too little bad

Kolbe Conative Connection

- Four action modes (MOs)
  - fact finder
  - follow through
  - quick start
  - implementor
- Three levels (continuum)
  - resistant
  - accommodating
  - insistent

Kolbe Conative Connection

- Fact Finder
  - insistent: must gather as much data as possible when starting a project, detail-oriented
  - resistant: eg: will use new computer for years without ever consulting manual or tutorial; may never discover many features
  - strength: researcher

Kolbe Conative Connection

- Follow Through
  - insistent: have to have a plan, focus on efficiency, organization, recognize patterns
  - resistant: anti-structure, anti-organization, eg: may clip coupons but never has on hand to use them
  - strength: designer

Kolbe Conative Connection

- Quick Start
  - insistent: begins projects with brainstorming, spontaneous and intuitive, risk and crisis-oriented
  - resistant: won't jump into water or new venture quickly
  - strength: innovator

Kolbe Conative Connection

- Implementor
  - insistent: hands-on, craft-oriented, prefers to deal with tangibles & the physical world
  - resistant: avoids getting hands dirty, lacks tools
  - strength: demonstrator

Kolbe Conative Connection

- Facilitator
  - no insistent mode
  - accommodating in all modes

- For more info:
  - http://www.kolbe.com
  - Kathy Kolbe, "The Conative Connection", 1990

Teamwork Day

HCI, UX, UI, oh my!

- HCI: human computer interaction
  - the study, planning and design of the interaction between people and computers
  - intersection of computer science, behavioral sciences, design
- UI: User Interface
  - the hardware and software systems which enable people to interact with machines
- UX: User Experience
  - how a user feels about using a system

(from Wikipedia)

HCI or UI?

- input modes
  - punch cards
  - keyboard
  - mouse, trackpad, trackball
  - touchscreen, stylus
  - voice
  - vision systems (Kinect)
  - motion
- output modes
  - print
  - visual
  - audio
  - tactile (vibrate)
- ease of use
- accessibility considerations

UX Design Goals

- user engagement
  - creating
  - maintaining
- rewards
  - user focus on goals, not tools
  - increased productivity
- levels of engagement
  - ease
  - pleasure
  - joyful immersion

(from Effective UI)

UX contributors

- science
  - human factors engineering
  - usability
  - information architecture (organization)
- art
  - design aesthetics
  - intuitive
- technical skills
  - implement design
  - underlying software quality

(from Effective UI)

UX necessities

- without science
  - confusing or frustrating to use
  - poor organization
- without art
  - ugly, uninteresting
  - users dislike, non-intuitive
- without technical skills
  - functionality is compromised
  - poor performance

(from Effective UI)

Elements of Engaging UX

- familiarity & consistency
  - QWERTY keyboard
- responsiveness & feedback
  - hourglass or progress indicators
- performance
  - processing time
  - reliability
- intuitiveness
- efficiency

(from Effective UI)

Constituencies

- Clients/Stakeholders
  - organization the product is being created for
- Users
  - people who will use the product
- Designers
  - those envisioning the systems
- Developers
  - those building the systems

(from Effective UI)

UX dichotomies

- idea vs. reality
- form vs. function
- design vs. implementation
- cost vs. time

Requirements vs. Features

- requirements specify product goals
  - usually dictated by the client
  - must be met
- features specify particular functions that may be used to achieve the goals
  - usually derived by the designers
  - primary and secondary priorities

(from Effective UI)

Case Study: CS Lab

- Clients: department admins
- Users: students (ugrads), TAs, faculty
- Designers: architects, support staff
- Developers: contractors, support staff
- Requirements: good computers to use, ability to create printouts, furniture, laptop space, office supplies, 24/7 access, allnighter accomodations, facilitate teamwork & brainstorming, nice ambience, social space
- Features: workstations- unix, easy access windows machines, good instructions for printer use, couch(es), vending machines, windows, whiteboards, collaboration space, extra monitors for dual input and/or bigger displays for laptops
- Good: 24/7 access, collaboration space, whiteboards
- Bad: lighting, spotty wireless, j-card swipe

Case Study: ISIS

- Clients:
- Users:
- Designers:
- Developers:
- Requirements:
- Features:
- Good:
- Bad:

Planning & Requirements

- uncertainty & unknown
  - givens – accept this reality
  - because design & construction must be integrated for success
  - consider two teams solving same problem
- change & subjectivity
  - requirements are a moving target
  - clients don't understand complexity of change requests

(from Effective UI)

The Process

- Deloitte Consulting guest lecture

Interaction Design (ID) - concept

- connecting people through the products they use
- more about behaviour than appearance
- new discipline unto itself (term coined 1990)
- a subset of user experience design
- interface design is the intersection between visual and interaction design

(from Designing for Interaction)

ID Elements

- focus on users
- create alternative solutions
- create uniquely appropriate solutions
- modelling & prototyping
- multidisciplinary influences
- collaboration
- incorporate emotion

(from Designing for Interaction)

ID Approaches

- user-centered design
- activity-centered design
- systems design
- genius design

- different situations calls for different approaches
- using more than one approach is often best

(from Designing for Interaction)

User Centered Design

- focus on user needs and goals
- involves the user at every stage:
  - requirements
  - functionality
  - features
  - prototyping
- requires extensive user research
- conative emphasis: fact finder

(from Designing for Interaction)

Activity Centered Design

- focus on tasks and activities (functionality and features)
- activities are actions and decisions (tasks) that serve a purpose
- more about creation than innovation
- research user behaviour
- conative emphasis: implementor

(from Designing for Interaction)

Systems Design

- focus on system components and interactions
- common system elements:
  - goals
  - environment (context)
  - feedback
  - regulation (action & reaction)
  - disturbances
- users set goals
- conative emphasis: follow-through

(from Designing for Interaction)

Genius Design

- "rapid expert" design
- designer innovates & creates
- entreprenuerial product, no clients/stakeholders, unidentified users
- no time
- user input for validation only
- conative emphasis: quickstart

(from Designing for Interaction)

The Process

- create design strategy (project proposal)
- conduct design research
- structured findings
- ideation and design principles
- refinement
- prototyping and testing

(from Designing for Interaction)

User Research Approaches

- interviews: in-depth, interactive, time-consuming, limited perspective
- surveys: cheap, anonymous, large-scale, inaccurate?, skewed results, what to ask?
- focus groups: immediate feedback, build on ideas, medium scale, peer influence, artificial environment
- observation: natural environment, see need in action, colored by actual instance of observation, hard to observe some aspects, intrusive?

(from Designing for Interaction)

Research Guidelines

- recruiting: choose diverse and appropriate user base
- scripting: have appropriate set of questions or bullet points to discuss
- conducting interviews:
  - pairs of researchers
  - go to users
  - talk, draw out stories & scenarios
  - take notes!
- ethics: get informed consent, maintain privacy, pay for subject's time

(from Designing for Interaction)

Creative User Input

- enact scenarios of use
- ask them to sketch ideas
- make models or collages
- directed story-telling
- user journals

(from Designing for Interaction)

Structuring Findings

- make data physical and visual
- use walls or large boards
- manipulate data points: clustering, combining, juxtaposing related, naming clusters, juxtaposing unrelated, sorting
- analysis tools: alignment diagram, touchpoint list, process map, task analysis
- build conceptual models: flows, sets, maps, diagrams, personas

(from Designing for Interaction)

Personas

- virtual representative users
- give fake picture & name to personalize
- based on user research
- focus on relevant characteristics that affect use of product:
  - behaviours
  - motivations
  - expectations
  - demographics only if relevant: age, gender, locale, etc.
- use in scenarios to evaluate product features, flow, etc.

(from Designing for Interaction)

Creating the Design

- brainstorming
- constraints – time, money, users, tools, technologies, skills
- identify design principles – key phrases that apply to multiple components
- direct vs. indirect manipulation
- affordances – physical/visual clues to functionality
- incorporate feedback (& feedforward?)
- standards

(from Designing for Interaction)

Frameworks

- structures that define the product and integrate the content and functionality into a unified whole
- many different forms: ie, metaphors
- site/screen/state maps most relevant to mobile apps

(from Designing for Interaction)

Design Documentation

- scenarios – stories of product use with personas as the characters
- sketches and models
- storyboards – narrated images
- task flows
- use cases – specific function explanations
- mood boards
- wireframes

(from Designing for Interaction)

Wireframes

- set of documents that show structure, information hierarchy, controls (navigation) & content (or placeholders)
- schematics or blueprints of a product
- mostly about features and functionality, not visual design
- should be drawn to scale for mobile devices
- usually accompanied by annotations and metadata (who, when, updates, etc.)

(from Designing for Interaction)

Prototypes

- don't need to represent entire system
- good to create multiple choices
- low-fidelity:
  - overall functionality and flow
  - may require manual manipulation to simulate interaction
  - paper – quick, editable, clearly drafts
- high-fidelity
  - look, feel, animation
  - aesthetics matter
  - should be interactive
  - the closer to finished product, the better the feedback on it
- do user testing with them!

(from Designing for Interaction)

Sensory Interface Design

- primarily concerned with feedback, layout and placement of controls
- natural flow – left to right, top to bottom
- position and alignment
- use to focus attention and create emphasis
- color can attract or detract
- contrasting fonts
- squint test

- sound effects
- touch (haptics)

(from Designing for Interaction)

A few Design Laws

- Fitt's Law: the larger or closer the target, the faster it can be reached (pointing)
  - size matters
  - corners and edges are infinitely large
  - co-locate controls with relevant data
- Hick's Law: decision speed is determined by the number of choices
  - two many categories or submenus is slow
  - better to have longer menus, within reason
- Magic Number Seven: humans best remember things in chunks of 5-9
- Tesler's Law of the Conservation of Complexity: some complexity is inherent in every process
- Poka-Yoke Principle: prevent inadvertent user errors

(from Designing for Interaction)

A few more design rules

- Ockham's Razor (KISS)
- 80/20 rule: 80% of apps use involves 20% of functionality
- satisficing = satisfying * sufficing
- progressive disclosure

(from The Resonant Interface)

Visual Design Considerations

- goal: efficient & intuitive use of app (& fun/appealing)
- elements:
  - color
  - icons
  - text
  - busy-ness, clutter
  - size (viewability, scalability, touchability)
  - brightness & contrast

Color

harsh new fresh	sunny, happy	neutral, soft
happy too intense energized	green on red is bad energy	dark
		neutral cute-sy?

GPA Class Design

- (see powerpoint slides)

Midtern

Android Activities

- one type of application component
- many apps will have multiple activities
- each activity has its own UI (screen)
- each activity has a lifecycle (review), connected to the android "back stack"
- each activity must be declared in the manifest as an element in that application
- each activity is defined in a corresponding java class file
- activities are activated by intents

Activity User Interfaces

- each UI is a hierarchy of ViewGroups & Views
- can be dynamically altered within java code using ViewGroup and View classes
- usually defined externally in a layout resource file (my_activity.xml) and inflated in code:

```
public class MyActivity extends Activity {
...
  public void onCreate(Bundle SavedState)
  {
    setContentView(R.layout.my_activity);
  }
```

- each layout xml file must have a root ViewGroup or View element
- each ViewGroup and View has a specific set of attributes

Android ViewGroups

- contain child View and ViewGroups
- each has set of LayoutParams that its children must define
- all must specify `layout_width` and `layout_height` attributes
  - `wrap_content`
  - `fill_parent` (`match_parent` in API Level 8)
  - if use explicit sizes, be relative to display size with density independent pixels (dp)

Android ViewGroup Layouts

- LinearLayout
- TableLayout
- RelativeLayout
- GridLayout
- Gallery
- FrameLayout
- ...

Declaring View Elements

- each should have an id attribute, which gets added to the R file if new
- declare in the layout (xml) for the activity in which it appears
- define corresponding object in the activity's class code & connect the two using the findViewById method:

```
Button save = (Button)
findViewById(R.my_activity.save_button)
;
```

Handling View Events

- UI views respond to user input events
- in java code, define event listeners and register them with the view
- handlers implement nested Listener interfaces (eg, `onClickListener`)
- define corresponding callback method (eg, `onClick`)
- register it to the View (eg, `setOnClickListener`)

Android Views & Widgets

- TextView
- EditText
- Button
- RadioButton
- CheckBox
- ViewFlipper
- Spinner
- ImageView
- ScrollView
- TabView
- ListView...

Android Menus

- Three types
  - Options menu: device menu key
  - Context menu: long click an item for floating list
  - Sub-menu: select a menu item that creates floater
- View hierarchies are built-in
- define callback methods
  - `onCreateOptionsMenu()`
  - `onCreateContextMenu()`
- handle their own events:
  - `onOptionsItemSelected()`
  - `onContextItemSelected()`
- can define menu items in an xml file

Team Design Presentations

Team work day

- (Joanne was too sick to lecture)

Versioning

- See Ken's slides on cvs

- Alternate system: Mercurial
  - install plug-in for Eclipse
  - BitBucket.org server to store files
  - gives GUI approach to updating

Android Data Storage - Overview

- various mechanisms available:
  - bundle to save activity instance state
  - SharedPreferences
  - external files (JAVA IO)
  - databases (SQLite)
  - content providers
- type used depends on
  - amount of data to store
  - sharing needs
- levels
  - activity: save activity state (user interface) when moved to background
  - application: share data between components
  - system:

Android Data: Saving Activity State

- why: to save and restore instance data as activity moves in & out of visibility
- what: data that is local to this activity
- how: use Bundle to saveInstanceState
  - this is a set of key,value pairs which is a special variation of Shared Preferences (next)
  - keys are unique strings
  - values are primitive types (including strings)
- saving: onSaveInstanceState

  only called when Activity becomes inactive, not when finish() or back button
- retrieving: onCreate, onRestoreInstanceState

Activity Bundle saving

```
@Override
public void onSaveInstanceState(Bundle myBundle) {
// get element whose data you want to save
// (may be instance data member already)
      Textview myText = (TextView) findViewById
                                    (R.id.myTextView);
// put it in your bundle
      myBundle.putString("Field_Key", myText.getText
                                          ().toString());
// do same with other elements
      myBundle.putFloat("Float_Key", myFloatInstanceData);

// call superclass method
      super.onSaveInstanceState(myBundle);
}
```

Activity Bundle restoring

```
@Override
public void onRestoreInstanceState(Bundle myBundle) {  // or onCreate
// call superclass method
        super.onRestoreInstanceState(myBundle);

// get element whose data you want to restore
        Textview myText = (TextView) findViewById(R.id.myTextView);
        String text = "";

// if bundle exists, get value from your bundle, second param is default if key doesn't exist
        if (myBundle != null)
        {       text = myBundle.getString("Field_Key", "");
                // do same with other elements
                myFloatInstanceData = myBundle.getFloat("Float_Key", 0.0f);
        }
// set your element
        myText.setText(text);
}
```

Android Data: Shared Preferences

- key,value pairs
  - keys are unique strings
  - values must be primitive types: boolean, string, float, long, integer
- used for
  - saving activity UI state (explicitly or implicitly with Bundles)
  - saving application settings
  - sharing data between activities in an app
  - storing user preferences
- use default application SharedPreferences object or give unique name to each preference file

Shared Preference objects

- use default for application:

```
Context context = getApplicationContext();
SharedPreferences myPrefs =
    PreferenceManager.getDefaultSharedPreferences(context)
    ;
```

- use private default for activity:

```
SharedPreferences myPrefs =
    getPreferences(Activity.MODE_PRIVATE);
```

- name your own:
  - create string prefFile = "MyAppPrefs";

```
SharedPreferences myPrefs =
    getSharedPreferences(prefFile,0);
```

  - second param is operating mode; 0=private...

Shared Preferences: storing

```
// assuming myPrefs SharedPreferences
   object has been initialized, need editor
SharedPreferences.Editor peditor =
   myPrefs.edit();
peditor.putBoolean("myFlag", true);
int count = 100;
peditor.putInt("myCount", count);
peditor.commit();   // TO SAVE CHANGES
```

Shared Preferences: retrieving

```
// assuming myPrefs SharedPreferences
    object has been initialized
// will use default values if key not found
boolean flag =
    myPrefs.getBoolean("myFlag", false);
int count = myPrefs.getInt
    ("myCount", 0);
String theString =
    myPrefs.getString("myString", "");
```

Android Data: External Files

- can use standard Java IO classes and methods
- Android specific methods for files in the current application folder only (filename is String):

```
// create output file private to this app
FileOutputStream fos = openFileOutput(filename,
    Context.MODE_PRIVATE);
// use Context.MODE_APPEND to avoid overwriting
    existing file for output

// create file input stream
FileInputStream fis = openFileInput(filename);
```

Android File Modes

- for external java File*Streams or SharedPreferences
- Activity.mode or Context.mode (App level)
- modes:
  - MODE_PRIVATE
  - MODE_WORLD_READABLE
  - MODE_WORLD_WRITEABLE
  - MODE_APPEND (OutputStreams)
  - MODE_MULTI_PROCESS (SharedPreferences)

Android: External File Resources

- can include in distribution package for app
- put in res/raw folder of project hierarchy
- myfilename does not include the extension
- use for large, pre-existing data sources

```
Resources myResources = getResources();
InputStream myFile = myResources.
    openRawResource(R.raw.myfilename);
```

Android Preference Activity Framework

- XML-driven framework available
- to create system-style preference screens
- enables consistency with native apps
  - familiarity is good!
  - can integrate settings screens from other apps
- three parts:
  - preference screen layout
  - preference activity
  - shared preference change listener

Preference Screen: Layout

- "prefs.xml" stored in the res/xml resources folder (not res/layout used for other UIs)
- use a special set of controls
- pref layout heirarchy starts w/PreferenceScreen

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
   xmlns:android:http://schemas.android.com/apk/res/android>
</PreferenceScreen>
```

- each PreferenceScreen is selectable element that will display new screen if clicked

Preference Screen: Layout

- within each preference screen include any combination of PreferenceCategory and Preference<control> elements
- PreferenceCategory elements break each screen into subcategories with a title bar separator
- Preference controls set the app preferences with attributes:
  - android:key - used to retrieve values from SharedPref
  - android:title
  - android:summary
  - android:defaultValue

Preference Category Example

```
<PreferenceCategory
   android:title="My Preference Category">
<CheckBoxPreference
   android:key="PREF_CHECK_BOX"
   android:title="Check Box Preference"
   android:summary="Small Type Descriptor"
   android:defaultValue="true"
   />
</PreferenceCategory>
```

Preference Controls

- CheckBoxPreference
- EditTextPreference
- ListPreference
- RingtonePreference

Preference Screen Options

- use built-in controls
- extend Preference class or built-in subclasses for customization
- use Intents to invoke screens from other apps:

```
<PreferenceScreen
   android:title="Intent preference"
   android:summary="imported system preference" >
   <intent android:action=
              "android.settings.DISPLAY_SETTINGS" />
</PreferenceScreen>
```

Preference Activity Class

- need class to extend PreferenceActivity
- connect activity to xml definition in onCreate:

```
public class MyPrefActivity extends PreferenceActivity {
//...
        public void onCreate(Bundle myBundle) {
          super.onCreate(myBundle);
          addPreferencesFromResource(R.xml.prefs);
        }
}
```

Preference Activity Manifest

- must declare activity in manifest file
- use Intent in manifest to make your preference available to other applications:

```
<activity android:name=".MyPrefActivity"
    android:label="My App Preferences">
    <intent-filter>
    <action android:name=
"joanne.cs250.myapp.ACTION_USER_PREFERENCE"
    />
    </intent-filter>
</activity>
```

Accessing Preference Values

- stored in SharedPreferences file for this Context

```
Context context = getApplicationContext();
SharedPreferences prefs = PreferenceManager.
                getDefaultSharedPreferences(context);
// use pref.get<type> methods to retrieve values
boolean check = prefs.getBoolean
                        ("PREF_CHECK_BOX", false);
```

Preference Change Listeners

- interface OnSharedPreferenceChangeListener
- lets application components listen for changes and make updates accordingly
- need to register the component to listen
- implement onSharedPreferenceChanged for each key that's relevant to this component

Preference Change Listeners

```
public class MyActivity extends Activity
        implements OnSharedPreferenceChangeListener {
/...
public void onCreate(Bundle myBundle)
{
    Context context = getApplicationContext();
    SharedPreferences prefs = PreferenceManager.
                        getDefaultSharedPreferences(context);
    prefs.registerOnSharedPreferenceChangeListener(this);
}

public void onSharedPreferenceChanged
        (SharedPreferences prefs, String key) {
    // check prefs values for key and update this as relevant
}
```

Databases

- see Yanif's database slides

GPA app

- multiple iterations
- lessons learned
- debugging in Eclipse
- critique

Sensors

- see Andreas's sensor slides and code for the fall detector app

Issues in Mobile Apps

- Code efficiency
- Responsiveness
- Seamlessness
- Compatibility

Code efficiency issues

- benchmark tools to measure
- pick best possible data structures
- memory at a premium
- avoid memory allocation
- processing at a premium
- in-line methods
  - avoid method calls
- avoid costly, superflous or redundant operations
- http://java.sun.com/docs/books/effective

Code efficiency issues

- use built-in libraries, often optimized
- tweaks
  - make constants static final if possible
  - make methods static if possible
  - access data directly, not setters and getters
  - use for each loop
  - use parallel arrays instead of arrays of objects, particularly for primitive data
  - 2D array processing – row order significantly than column order
  - eliminate 2D arrays

Responsiveness issues

- acceptable response time: 100-200 ms
- ANR – application not responding
  - delayed response to user input (5 sec)
  - other slow events (Android broadcast receivers 10 sec)
- slow operations:
  - database operations
  - data intensive computing – bitmap resizing, game move calculations, digital signal processing
  - network operations

Responsiveness issues

- use child threads for slow operations
- add handler for thread returns
- minimize ops in onCreate and onResume
- use progress indicators
- make dancing bears while app loads
- use services & notifications instead of broadcast receivers
- battery drains: processor & radio use

Seamlessness issues

- screen consistency within app
- consistency w/platform apps
  - extend built-in themes
- app running in live environment
  - other apps interrupt – consider system interactions
  - background processes use notifications not dialogs
  - use activity lifecycle methods appropriately
    - onSaveInstanceState, onPause
    - particularly for edit screens

Seamlessness issues

- use ContentProviders instead of world readable raw data files or databases if sharing
- use multiple activity screens

Compatibility issues

- screens
  - size (dimension) – declare for small screens
  - density
    - use dp (dip), sp (sip)
- landscape mode
- hardware features
- API levels: declare min-sdk, target-sdk
- internationalization

Deployment Preparation

- clean-up code
- aggressive testing
- set appropriate min-sdk, target-sdk, uses-features
- documentation
- release to test users

Cleaning code

- eliminate unnecessary/unused components
- refactoring (Eclipse tools – GPA example)
- use optimizer tools
  - android layout optimizer
  - android zipalign to optimize final apk
- straighten out repositories

Project Presentations

- Fix My Spill
- Yahoo Answers
- SoundTrak
- Personal Pharmacist

Project Presentation

- Wep Cracker

Testing – how to

- Android testing framework extends JUnit
  - see tutorials & documentation
- UI/Application Exerciser Monkey:
  - referred to as "monkey"
  - command-line tool to send random streams of user input to a device
  - run it with adb tool
- monkeyrunner tool: an API and execution environment for test programs written in Python

Testing – what to

- all activity components
- lifecycle based methods
- helper methods
- input validation
- changes in orientation & device configuration
  - default behavior is to destroy & restart foreground activity
  - is UI redrawn correctly?
  - does the app maintain its state?
- battery usage
- dependence on external resources

Next Steps

- See Ken's slides on marketing your app

Societal Context

- Challenges
  - security & privacy
  - network connectivity
  - power consumption
- Target Areas
  - healthcare
  - developing countries
- HCI/UI advances
  - disabilities

Wrap-up

- Thank-you!
- Course Evaluations
- Phone Return – by 5/18!