http://www.cs.jhu.edu/~joanne/cs250

Joanne Selinski
Spring 2017

600.250
User Interfaces and Mobile Applications
Course Overview

1st Case Study: Hi-Lo Game App

DAY 1 – 2/1/2017
Course Components

 User Interface Design
 Mobile Application Development
  • front-end implementations
  • back-end lifecycle, navigation and data management
 Soft Skills
  • teamwork skills
  • communication
  • leadership
  • critiquing – giving and receiving
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
  - GUI: graphical user interface

- **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
[Why] Is Design Important?

Small -> larger group discussions...
[Why] Is Design Important?

some ideas...

- design vs. functionality
- usability is the intersection?
- can't use feature if can't find or work it
- won't use app at all if it doesn't work or have decent features

Conclusions:
- yes, but not at the expense of functionality
- integral part of app ideation & creation from start
HiLo Game – Design Phase

- Functional elements:
  - pick a random number
  - take user input
  - compare numbers
  - display result
  - repeat until correct
  - reset & play again

- Design elements:
  - text user input
  - check input action/button
  - hi lo indicator display
  - game over display
  - mechanism to replay
What is Android? (your ideas)
What is Android?

- operating system
- linux based
- optimized for mobile
- platform independent
- open source
- play store
- framework, API
- for phones, tablets, wearables, etc.
- google
- evolving design elements
- customizable
Why Android?

- can develop on anything
- java based
- free to develop
- open/flexible
- have (old) phones & (new) tablets for you
- many devices
- biggest and best
- job skill
- easily shared
How Android?

- Android Studio
- Emulators
  - within Android Studio (slow)
  - GenyMotion (better, requires VirtualBox)
- Code Samples
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.
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**
  - user events trigger pre-determined method handlers
HiLo Game: Implementation

- brief overview of a very rudimentary version in Android Studio...
UX Design Laws
Exercise: Design An App For ??

DAY 2 – 2/3/2016
User Interfaces – your ideas

GOOD
intuitive
visual indicators
simplicity
fun
polished
progress indicator
symmetry

BAD
too much functionality
too much clutter
"ugly" color schemes
too hard to read/see
inconsistency
too many clicks
distracting
User Interfaces – (other) ideas

**GOOD**
- intuitive to use
- clean, simple, elegant
- right level of information
- fast response
- minimal overhead
- easy to learn
- consistent behaviour
- consistent look & feel
- logical organization

**BAD**
- too much customization
- no customization
- buggy features
- dated or poor aesthetics
- disorganized features
- hidden features
- overcrowded screens
- too complicated
- lack of response, feedback
- pop-ups
- physically unwieldy
UI/UX references

- **Designing with the Mind in Mind: Simple Guide to Understand User Interface Design Rules**, Jeff Johnson (MSEL on-line)
- **The Design of Everyday Things**, Donald Norman, [http://www.jnd.org](http://www.jnd.org)
- **UX Research & Training**, Jakob Nielsen, [http://www.nngroup.com](http://www.nngroup.com)
- **Designing for Interaction**, Dan Saffer (MSEL on-line)
- **Effective UI**, Jonathan Anderson (MSEL on-line)
Brief History of Design Rules

- Cheriton, 1976 - for early interactive computer systems
- **Norman**, 1983, software UI design rules based on human cognition
- Smith and Mosier, 1986, comprehensive
- **Shneiderman**, 1987, "Eight Golden Rules of Interface Design"
- Brown, 1988, HCI design guidelines
- **Nielsen** & Molich, 1990, heuristic evaluation of user interfaces
- Marcus, 1992, graphic design guidelines

(from Designing with the Mind in Mind)
Video

- Donald Norman's TED talk on design & emotion:
  http://embed.ted.com/talks/don_norman_on_design_and_emotion.html
- (13 minutes)
Shneiderman's Design Rules

- strive for consistency
- cater to universal usability
- offer informative feedback
- design task flows to yield closure
- prevent errors
- permit easy reversal of actions
- make users feel they are in control
- minimize short-term memory load

(from Designing with the Mind in Mind)
UX dichotomies

- idea vs. reality
- form vs. function
- design vs. implementation
- cost vs. time
- bright screen vs. long battery life
- powerful & simple
- ...
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
  - progress indicators
- performance
  - processing time
  - reliability
- intuitiveness
- efficiency
- attractive
- fun (touch of whimsy)

(from Effective UI)
Small Group Activity

- Design an app to "practice random acts of kindness and senseless acts of beauty"
  - allow random or user choice of recipient from contacts
  - give choice of sending text or photo
  - if text – allow custom message or random selection from built-in phrases
  - if photo – option to take or choose from gallery
- Choice of paper drawing or electronic mock-up
Android Components
Layouts And Common View Elements
Case Study: Hi-lo Improvements

DAY 3 – 2/8/2016
Android Software Stack

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

- set of core libraries
- multi-user Linux system
- each app acts as a user on the system
- virtual machine
  - an instance for each application
  - unique ID assigned to each
  - optimized for concurrently running instances
  - principle of least privilege
  - sharing between apps is possible when necessary
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 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)
Android Hardware Features*

- Optimized graphics
- Media support for audio, video, and still images
- GSM Telephony (Global System for Mobile Communications)
- 3G/4G
- WiFi
- Bluetooth
- 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**
Application Examples

- **foreground app: calendar**
  - activities: day, week, month views, create event, choose time, reminders
  - 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**
HiLo Game: Implementation

- download code: http://cs.jhu.edu/~joanne/cs250/hiloBad.zip
- run it to see what is wrong
- let's review the overall project structure and element details...
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
  - user events trigger pre-determined method handlers
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.
Activity User Interfaces (Layouts)

- 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:

```java
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
- for best performance, avoid many nested levels of ViewGroups
- each has set of LayoutParams that its children must define
- all must specify `layout_width` and `layout_height` attributes
  - `wrap_content`
  - `match_parent` (fill_parent in API Level <8)
  - if use explicit sizes, be relative to display size with density independent pixels (dp)
Common Layouts

- **ViewGroups**
  - RelativeLayout – flexible, but can be tricky to get right
  - LinearLayout – horizontal or vertical ordering
  - TableLayout
    - RowLayout
  - GridLayout – adaptively populated at runtime
  - FrameLayout – usually dynamic

- **AdapterViews**
  - to dynamically populate views at runtime
  - GridView
  - ListView
RelativeLayout

- define children relative to each other and the parent view
- use view IDs to reference each other
- child view attributes for example:
  - above, below
  - alignParentRight, alignParentTop
  - centerVertical, center (horizontal)
LinearLayout

- aligns children vertically or horizontally according to orientation attribute
- weights are used to distribute children in the space provided
  - by default weight is 0 if unspecified, given enough space to display content
  - higher weights occupy a larger percentage of space
  - children with equal weights are same size
  - eg: 2, 1, 1 for 3 children – 50%, 25%, 25%
Common Views & Widgets

- TextView
- EditText
- Button
- RadioButton
- CheckBox
- Spinner
- Picker
- ImageView
- ScrollView
- ...
Declaring View Elements

- each should have an id attribute, which gets added to the R file if new (use @+id)
- 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:

```java
Button save = (Button)
findViewByById(R.my_activity.save_button);
```
View Properties

- View elements have many properties that allow them to be customized. These include:
  - appearance – dimensions, layout, text size, fonts, colors, etc
  - behaviors – responding to user actions such as click, touch
  - focus & visibility
  - input type (particularly for EditText fields)
Handling View Events

- UI views respond to user input events
- In Java code, we define event listeners and register them with the view
- Handlers implement nested Listener interfaces (e.g., `OnClickListener`)
- Define corresponding callback method (e.g., `onClick`)
- Register it to the View (e.g., `setOnClickListener`)
Button Handling Options

 define method in activity, set in layout
  • method must be public, void, with View param

 define Listener and Handler together:
  Button button = (Button) findViewById(R.id.button_send);
  button.setOnClickListener(new View.OnClickListener()
  {
    public void onClick(View v)
    {
      // Do something in response to button click
    }
  });

 example: DeStress app – has one of each type
Alternate Button coding style

- Declare button and set listener:
  
  ```java
  Button mybtn = (Button) findViewById(R.id.myBtn);
  mybtn.setOnClickListener(myListener);
  ```

- where myListener is defined as a private member object of the class:
  
  ```java
  private OnClickListener myListener = new OnClickListener() {
      @Override
      public void onClick(View v) {
          // do stuff
      }
  };
  ```
EditText Handlers

- We can respond to text input in multiple ways:
  - give the user an explicit button to click
  - specify behavior for the action button on a soft input keyboard
    - in layout: android:imeOptions="actionSomething"
    - in code: editTextField.setOnEditorActionListener(…)
  - create a TextWatcher to respond to a text change event
- In designing our apps we should choose the method that gives the user the best experience...
Hi-Lo Game Improvements

- update the number keypad so that the bottom right corner says "GUESS"
- fix it so that the input field is cleared after each guess
- add a button to reset the game, include the back-end processing
- change the input to use a seek bar – what else about the layout should change?
- what other improvements to the appearance should we make?
Design Process I
Teamwork
Practical Design Rules
Design Critiques I

DAY 4 - 2/10/2017
Mobile App Dev Process

- make teams
- brainstorm project ideas & pick one
- research: existing, marketability – users?
- design – features, UI
- implement – front end, back end
- testing
- marketing
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)
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)
Example: CS Lab

- Clients: department staff
- Users: students (ugrads), TAs, faculty?
- Designers: architects, support staff
- Developers: contractors, support staff
- Requirements: good computers to use, ability to create printouts, comfy seats, laptop space, office supplies, 24/7 access, allnighter accommodations, 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
Case Study: Landscape App

- **Clients:** Landscaping company management
- **Users:** employee landscapers, mowers, etc.
- **Designers:** us
- **Developers:** us
- **Requirements:** record job, bill job, keep track of customers, keep track of employees
- **Features:** create/edit customer info, create/edit employee info, employee login, job record, email billing, etc.
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)
Teamwork – your ideas

**GOOD**
- version control
- communication
- using strengths
- trust
- deadlines – making, keeping
- helping teammates
- working physically together, paired programming
- open minded

**BAD**
- laziness, freeloading
- not doing what you say you will
- micromanaging, dictator
- doing for other without teaching
- passive attitude
- not asking for help when needed
Teamwork – Communication I

GOOD
- suggestions
- be specific
- frequent
- feedback
- compliment sandwich
- pose dislikes with alternative options

BAD
- fugly, sucks
- non-constructive insults
- too many channels
- condenscending
<table>
<thead>
<tr>
<th>GOOD</th>
<th>BAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;needs more ...&quot;</td>
<td>&quot;that sucks&quot;</td>
</tr>
<tr>
<td>&quot;too ...&quot;</td>
<td>&quot;start over&quot;</td>
</tr>
<tr>
<td>&quot;personally doesn't work for me&quot;</td>
<td>&quot;i hate it&quot;</td>
</tr>
<tr>
<td>&quot;would be better if...&quot;</td>
<td>being wishy-washy</td>
</tr>
<tr>
<td>&quot;what were you thinking when you...&quot;</td>
<td>not voicing opinion</td>
</tr>
<tr>
<td>politely direct</td>
<td>being dismissive</td>
</tr>
<tr>
<td>use examples</td>
<td>laughing</td>
</tr>
<tr>
<td>be open to</td>
<td>voicing false opinion</td>
</tr>
<tr>
<td>suggestions/critique, don't get defensive</td>
<td>dominating</td>
</tr>
<tr>
<td>ask questions</td>
<td></td>
</tr>
</tbody>
</table>
Individual Responsibilities

- amount of work
- living up to commitments
- on time
- communication levels
- asking for help when needed
- attend meetings
- be available
- courteous & accommodating
- bringing ideas
- overall effort
- helping others
- taking initiative
- listening to others ideas
- planning

- being respectful
- staying focused in meetings
- stepping up when there is need
- giving good feedback
- giving input to all aspects
- receiving unsolicited help
- accepting of feedback
- organization
- quality of work
- only take on what you can handle
Teamwork - Diversity

- skills
  - planning
  - research
  - designing
  - implementing
  - promoting
- personalities
  - leadership
- backgrounds?
- friendship – too much bad, too little bad
Teamwork Process Essentials

- create realistic requirements
- decomposition of project components
- deadlines should be set and include buffer time
- integration takes longer than you think

(from Effective UI)
Paired Programming

- two roles:
  - Driver: hands-on keyboard
  - Navigator: eyes on screen and future

- simple guidelines:
  - change roles periodically
  - talk lots
  - listen lots
  - teach each other
  - ask for help
Some Practical 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

(from Designing for Interaction)
A few more design laws

- **Ockham's Razor**: (KISS) keep it simple
- **80/20 rule**: 80% of apps use involves 20% of functionality
- **Poka-Yoke Principle**: prevent inadvertent user errors
- satisficing = satisfying * sufficing
- progressive disclosure

(from The Resonant Interface)
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)
Visual Design Considerations

- **goal:** efficient & intuitive use of app (& fun/appealing)

- **elements:**
  - color
  - icons
  - text
  - busy-ness, clutter
  - size (viewability, scalability, touchability)
  - brightness & contrast
Design Critique Guidelines

- It’s Not About You
- Maintain Respect and Honesty
- Avoid Meaningless Words and Phrases
- Comparisons Should be Used Sparingly
- Be Specific
- Remember those Design Principles
- Understand the Design Approach and Context
- Ask Why
- Offer Suggestions
- Consider the Goals and Audience of the Design
- [http://tympanus.net/codrops/2012/10/15/the-unwritten-rules-of-a-great-design-critique/](http://tympanus.net/codrops/2012/10/15/the-unwritten-rules-of-a-great-design-critique/)
Exercise

- Evaluate apps to "practice random acts of kindness and senseless acts of beauty"
  - allow random or user choice of recipient from contacts
  - give choice of sending text or photo
  - if text – allow custom message or random selection from built-in phrases
  - if photo – option to take or choose from gallery
- Critique rak designs as assigned (http://cs.jhu.edu/~joanne/cs250)
- Do reviews in pairs
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 Lifecycle

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

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

- **components**
  - java class in source file: MyActivity.java
  - public class MyActivity extends Activity
  - layout file in res: my_activity.xml
  - declaration in manifest:
    ```xml
    <activity android:name=".MyActivity"
               android:label="@string/my_activity_title">
    </activity>
    ```

- **main activity settings in manifest**:
  ```xml
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
  ```
Case Study: Destress

- Destress (hit me) zip file posted
- Illustrates difference between back button and action bar return
- Uses context shared preferences to store data at various points
Android Intents

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

- We create an intent to move from the current activity (ActOne) to some other activity (ActTwo):
  ```java
  Intent intent = new Intent(ActOne.this, ActTwo.class);
  ```

- We launch ActTwo with this intent:
  ```java
  startActivity (intent);
  ```
Returning from Activities

- Call `finish()`; to end the current activity and return to where it was launched!

- To specify where to return when using the back navigation button in the action bar from this activity, explicitly define parent activity in manifest where child is defined:

```xml
<activity
    android:name=".ChildAct"
    android:parentActivityName=".Parent"
    android:label="@string/title_child">
</activity>
```
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: share data among different apps
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
  - 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);
    // save updates
    myBundle.commit();
    // call superclass method
    super.onSaveInstanceState(myBundle);
}
Activity Bundle restoring

@Override
public void onRestoreInstanceState(Bundle myBundle) { // or in 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 the application's default 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:
  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 (2\textsuperscript{nd} param) 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):

```java
// 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

```java
Resources myResources = getResources();
InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```
Design Process II
Design Critiques II
Quiz I

DAY 6 – 2/17/2017
The Design Process

- create design strategy (project proposal)
- conduct user research
- structure 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 these 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 or themes — 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)
a1B designs were done in pairs

go to http://cs.jhu.edu/~joanne/cs250/a1B

reviews.txt says who reviews what

individual reviews (15 min) (2 per design)

group reviews – 6 people, 3-4 designs (10 min)

class discussion – 10 groups (10 min)
Quiz 1

- On Blackboard
- Timed: 10 minutes
- Password is: easyOne
Navigation Types

 How many navigation elements or component options can we name?

 What are good examples of android apps that use them?
Our List of Nav components

- action bar
  - menu (overflow)
- swiping
- button (activating an intent)
- navigation drawer
- short click on View element
- long press on element
- timed activity duration
- voice?
Demos

- Yahoo weather (nav pane, scrolling, +)
- Slack (action bar, nav pane, scrolling, drop-down, menus)
- Calendar (nav pane, action bar, scrolling, listview, menus)
- Clock (tabs, action bar, swipe)
- Messages (listview, action bar, floating action button)
- SHealth (gridview, tabs, action bar, menus, drop-up)
Planning Screens & Relationships

- Identify the primary entities and their interactions
  • can be done with an ERD – entity relationship diagram

- Make a screen list
  • exhaustive set of screens to allow user to interactive with and navigate the app

- Create a screen relationship diagram
  • clarifies which screens should lead directly into others, and where to return
Multiple Device Considerations

- small devices typically show 1 screen at a time
- larger devices should use multi-pane layouts
- apps can provide a variety of layout options (device size & orientation) for maximum use and flexibility
Navigation Types

- **Descendent**
  - go from parent screen to child screen
  - example: from list of items to one particular item detail

- **Lateral**
  - go from one sibling screen to another
  - collection siblings are multiple items in a collection (such as items in a list)
  - section siblings are used to present different sets of information about a parent

- **Ancestral**
  - provide ways to easily go up/back and home
Navigation Diagram

Collection Siblings ➔

Section Siblings ➔
Navigation & Layout Options

- listview, gridview – good for collection siblings
- tabs, swipe – good for section siblings
- buttons – good from parent to section children (dashboard pattern)
- multi-pane – best implemented with combination of activities and fragments for reuse
- navigation drawer
- menus
Navigation Drawer

- Use for high level navigation to various activities within app

Android Menu Types

Three types

- Options menu:
  - overflow menu icon in action bar (newer phones)
  - device menu key (old phones)
- Context menu or contextual action mode: long click an item for floating list or bar of actions, usually applied to item(s) in ListView or GridView
- Pop-up/sub-menus: anchored to specific view elements

View hierarchies are built-in

can define menu items in an xml file (preferred) or java code
Options Menus

- appears in action bar in modern Android
- items can be displayed in action bar if room, or always relegated to the hidden menu
- sometimes can add overflow menu to bottom bar
Context Menus

- long click an item for floating list
- typically used for items in ListView or GridView
- define menu items in an xml file
- register the view associated with menu:
  - registerForContextMenu(myListView)
- this was the only type of context menu available in older android versions
Menu Handlers

- define callback methods
  - onCreateOptionsMenu()
  - onCreateContextMenu()

- handle their own events:
  - onOptionsItemSelected()
  - onContextItemSelected()
Contextual Action Mode

- long click presents a contextual action bar to the user
- can enable selection of one or multiple items
- action bar task chosen will be applied to the selected items
- system implementation of ActionMode
  - (more details: https://developer.android.com/guide/topics/ui/menus.html#context-menu)
Navigation Drawer vs. Options
Menu vs. Tabs

- when to use each?
- where is duplication good?
- when is duplication bad?
Case Study: Landscape App

- Suppose we want to design an app for a landscaper to use to keep track of his/her business.

- Entities: user=landscaper, customers, central office, equipment, employees, cash flow, inventory, suppliers

- Main Features: job scheduling, tracking project & needed resources, billing clients, access location information,

- Secondary Features: calendar integration, tracking & updating employee information, assigning workers to a project, notification job reminders, tracking equipment & maintenance, checking balances/receipts/cash
Implementing Information Flow: Navigation Drawer, Menus Fragments
Landscape Design Exercise

- Let's make a feature list
- Let's make a screen list and relationship diagram
- Let's figure out what it would take to implement
- (small group exercise, large group review)
Feature/Screen List

- **Primary**
  - login, profile update
  - add/edit new job
    - where, when, bees, client
    - supplies, services, total cost
    - bill the client
  - job log: view, edit/update, delete

- **Secondary**
  - maintain client list
  - budgeting/expenditures
  - maintain employee list
YardPro App Demo – v1

- Navigation drawer
- Options menu content is created in xml
- Context menu content is created in code
- Uses ListView for jobs
  - backend database we'll get to later

- Illustrates launching activities with results
- Illustrates putting extra data in Intents
Nav Drawer implementation

- NavUIExample – basic navigation/action bar new project from Android Studio
- Hierarchy of layouts, use of `<include>`
  - `activity_main` is `DrawerLayout`
    - `<app_bar_main` is `CoordinatorLayout`>
      - `app_bar_layout` (action bar looks)
      - `<content_main` is `RelativeLayout` (actual content)>
      - floating action button
    - `NavigationView` (the nav drawer)
      - `nav_header_main` (header layout)
      - menu: `activity_main_drawer` (actual items)
  - menu: `main` is options menu items, inflated in `MainActivity.java`
Navigation, Activities & Fragments

- Navigation Drawer is component of an Activity
- Action Bar is defined per an Activity (options menu)
- In order to have several views share a nav drawer and action bar:
  - define a container activity for them
  - put a fragment container as the main view of the activity
  - define each different view as a fragment, not an activity
  - bind the fragments to the main activity
  - swap out & inflate each fragment depending on the navigation items chosen
Fragment Basics

- A Fragment is a subdivision of an Activity
- Must be attached to an Activity, and needs an Activity to run
- Can be reused in multiple activities
- The advantage of building applications that use Fragments is their ability to easily present a consistent well-designed user interface
- Fragments are particularly helpful in adapting a user experience across a wide range of devices with varying size and layout configurations
Fragment Lifecycle

- Fragments are associated with an activity and can be created, added or removed while the activity is running.
- Similar to an Activity, a Fragment has its own lifecycle, as well as its own user interface.
- The Fragment’s lifecycle is connected to the activity that owns it.
- Each fragment has its own callback methods in the standard Activity lifecycle, as well as special Fragment callback methods.
Fragment Lifecycle methods

- `onCreateView()` - use to inflate the fragment and return its corresponding xml layout
- `onInflate()`
- `onActivityCreated()`
- `onAttach()`
- `onDestroyView()`
- `onDetach()`
How to use Fragments

- A user interface can be divided into multiple panes using Fragments and reused in more than one screen of an application.
- Responsive design makes use of Fragments to provide adaptive views and alternative layouts for different configurations.
Implementing Fragments

- unlike activities, they are not declared in the app's manifest
- Java class must extend Fragment
  - similar to an Activity, put the lifecycle handling methods here
- instantiate a Fragment object from the class definition, usually within some activity
- the fragment object must be bound to an activity
- can use a fragment container tag in the xml layout for the activity that will use it
Fragment Example

- Code example (posted)
- Creates main activity with nav drawer and action bar
- Swaps between two fragments within the main activity
- Nav drawer menu items are used to access the fragments
Back-end Data:
Intents with data & results
ListView
SQLite Database
Case Study: Landscape App
YardPro App Details – v1

- Navigation drawer
- Options menu content is created in xml
- Context menu content is created in code
- Illustrates launching activities with results
- Illustrates putting extra data in Intents
- Uses ListView for jobs
- Backend database (SQLite)
Sharing data between activities: Intents with Data

- Including data in an intent can be done with `<key,value>` pairs, where the keys are strings:
  ```java
  Intent intent = new Intent(ActOne.this, ActTwo.class);
  intent.putExtra("some boolean", false);
  intent.putExtra("some string", "message in a bottle");
  ```

- Getting the intent that launched this activity:
  ```java
  [Intent] launcher = getIntent();
  ```

- Getting data out of the intent – second param is default value to use if the key isn't there:
  ```java
  launcher.getBooleanExtra("some boolean", false);
  launcher.getStringExtra("some string", "no msg");
  ```

- You can put and get a variety of basic data types
Activities with Results - I

- we want ActOne to initiate ActTwo, but get a result back from ActTwo
- define corresponding request code in ActOne:
  
  ```java
  static final int REQUEST_CODE = 1;
  ```
- as a response to a user event in ActOne:
  ```java
  Intent intent = new Intent(ActOne.this, ActTwo.class);
  startActivityForResult(intent, REQUEST_CODE);
  ```
- now handle what to do when ActTwo comes back to ActOne:
  ```java
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
      if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
          // code to execute when you return from ActTwo successfully
      }
  }
  ```
Activities with Results - II

- In ActTwo we need to set results before returning to ActOne:

  ```java
  setResult(RESULT_OK);
  
  or

  setResult(RESULT_CANCELED);
  ``

- and then explicitly go back to where this was launched:

  ```java
  finish();
  ```
Passing data to/from Fragments

- Tricker than bundling data in intents when launching activities
- Can use application or activity level SharedPreferences file(s)
- May need to access the containing activity explicitly, usually done in `onCreateView()` for the fragment:
  ```java
  MyActivity holder = (MyActivity) getActivity();
  ```
- There are a number of ways to go from here, and other approaches as well
ListView Basics

- A ListView is a specialized control that is optimized for displaying long lists of items.
- When the data content for the layout is dynamic or not pre-determined, it is possible to use a layout that subclasses an AdapterView to populate the layout with views at runtime.
- We can also create a custom item adapter for displaying the items in a ListView.
- When the data content is static (unchanging), you can just create the list items in arrays.xml instead.
ListView Actions

- A ListView might be populated with items that need to respond to a (single) click event.
- You can respond to click events on an item in an AdapterView by implementing the AdapterView.OnItemClickListener interface.
- The onItemClick() callback method will always be invoked when an item in the AdapterView has been clicked.
- Items can be "registered" for a ContextMenu as well, which is activated by a long click.
Context Menus for List Items

- use an xml file to create context menu for the list items (with actions such as Edit, Delete, etc)
- register the listview for context menu in onCreate
- inflate the menu in onCreateContextMenu
- specify how to respond to menu selections in onContextItemSelected
Android Data Storage - Review

- 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: share data among different apps
Database Concepts

- **A relational database** is a collection of tables, each of which specifies a relationship among various data elements.
- Tables are compose of **records** (rows), each of which has a unique ID called a **key**.
- The columns each represent a different **attribute** of the record, and the column headers form the **schema** of a relation.
- Tables may have different components of related elements, such as student degree information and student contact information, which share the student ID in common.
Relational Database Tables

<table>
<thead>
<tr>
<th>JHED (key)</th>
<th>First Name</th>
<th>Last Name</th>
<th>Phone</th>
<th>Birthdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsmit2</td>
<td>Fred</td>
<td>Smith</td>
<td>323-234-2345</td>
<td>09/06/1995</td>
</tr>
<tr>
<td>mjone3</td>
<td>Mary</td>
<td>Jones</td>
<td>235-234-2345</td>
<td>03/14/1997</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JHED (key)</th>
<th>Major</th>
<th>Degree</th>
<th>Start Year</th>
<th>Degree Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsmit2</td>
<td>Computer Science</td>
<td>BS</td>
<td>Fall 2013</td>
<td>May 2017</td>
</tr>
<tr>
<td>mjone3</td>
<td>Applied Math</td>
<td>BS</td>
<td>Fall 2012</td>
<td>December 2015</td>
</tr>
</tbody>
</table>

...
Database Operations

- We can perform operations on database tables in order to form new relations:
  - **select**: select a subset of records (rows) based on particular criteria (such as major or degree date)
  - **project**: take a subset of attributes (columns) and optionally perform operations on them (summing, average, etc)
  - **join**: combine data from multiple relations based on common fields

- This is a sampling of operations, and many combinations are possible also.
Databases in Android

- SQLite is a slimmed down version of SQL, Structured Query Language, for performing operations on relational databases.
- Core Database class: SQLiteOpenHelper
- Database Adapter class: recommend to write one to make handling the SQL operations easier from the app activities.
- Example: YardPro1.zip
SQLite in a DBAdapter

- We define the attributes (columns) we want our records to have.
- We create tables of records with specific columns of interest.
- Each time we define (or redefine) a table in a database, we need to change the database version number so it gets updated properly.
- ContentValues is an object that holds the data for one record.
- We use a Cursor (like an iterator) to move through the records in a database table (or resulting from a query).
- See JobDBAdapter.java.
Databases & ListViews

- typically the items in a listview correspond to records in a database
- there is an intermediate layer of a class object connecting the two (JobItem)
  - the toString may be used to populate what gets displayed about an object in the list item layout
- the class objects are collected into an ArrayList
- some form of ArrayAdapter is used to connect the ArrayList of objects to the ListView (JobItemAdapter)
- the position in the Listview is used to find the record in the database
Implementing common actions

- first find out which item is chosen based on the item ID
- get the corresponding record from the database
- make the changes to the record (see next slide)
- update the database
- update the listview and redisplay

- delete: just get item ID and delete corresponding record
Edit item actions

- edit more complex
  - may need to populate an item view with existing data
  - can use extras bundled into Intent to pass current data to an edit activity
  - may need to track whether adding item or edit item if the same detail activity is used for both
  - need to update individual fields in database

- if same activity used for edit & add an item – different save actions
  - check intent to see how it was called
  - update and finish (return) if in edit mode
More about Databases

- See Prof. Ahmad's slides: http://cs.jhu.edu/~joanne/cs250/databases.pdf
- Take Prof. Yarowsky's course 600.315 Intro Databases
Design Critiques
Quiz II

DAY 10 – 3/3/2017
Design 2 peer reviews

- see reviews.txt for what to review and groups
- do individual reviews based on assignments
- group discussion of the designs reviewed
- class discussion of best per group
Quiz 2

- On Blackboard
- Login, will have 10 minutes to do
- Password is:
Teamwork Day
Joanne is away
meet in classroom or BLC or CS Lab or …

DAY 11 – 3/8/2017
versioning systems

git overview

setting up repos on BitBucket

integration with Android Studio

DAY 12 - 3/10/2017
Teamwork – Coding Techniques

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

- Facilitate multiple developers working on the same project
  - each has cloned local source tree (file structure)
  - allows notification of conflicting changes, merging of work
- Maintain history of all changes, along with documentation on them
- Manage multiple development branches including patches and major releases
- Can revert to previous versions if newer "upgrades" end up being worse
Versioning Systems - How

- central database of project
  - stores all source code
  - stores complete history of changes: branching information, what/who/why
  - allows any past version to be reconstructed
- source code is cloned as local copies
- developers check repository for updates to local copy, make updates, check again, resolve conflicts, commit changes to shared repository
Versioning Best Practices I

- Commit changes reasonably frequently: merge issues may become complex if you wait too long.
- Always compile and test before committing updates: your team mates will hate you if they update and things no longer compile!
- Things can become complex if multiple teams share common repositories: make a separate one for each project.
Versioning Best Practices II

- Avoid putting proprietary file types in repos (i.e., `.docx`, `.pptx`, etc.): they can't be diffed easily.

- Agree on common coding style
  - You don't want multiple versions because of spacing changes for example.
  - Always stylecheck before committing!
  - (Eclipse can help with this a lot)

- Stash changes (save copy) before pull in case automerge messes things up.
Versioning Systems - What

- Many version control varieties out there:
  - cvs
  - subversion (svn)
  - mercurial
  - git
- Main repository can be stored on a central service:
  - bitbucket.org – can be used with git or mercurial
  - github.com – mostly for open source
- Command-line usage
- Some integration with IDEs
Setting up BitBucket

- go to bitbucket.org – everyone should create a personal account
- one person on each project team should create a team on their bitbucket account
- add other team members to the team using their bitbucket accounts, setting all as administrators
- one person can then create a Team Repository
Versioning w/git

- Snapshot of common usage: [http://cs.jhu.edu/~joanne/git.html](http://cs.jhu.edu/~joanne/git.html)
- Git tutorial by Lars Vogel: [http://www.vogella.com/articles/Git/article.html](http://www.vogella.com/articles/Git/article.html) (good overview, also command-line usage)
Git Basics

- **clone** repository – only need to do this once to create a local copy
- **pull** (--rebase) to update your local copy from the central version before each and every work session (combination fetch & merge)
- make changes, create files, etc. – only affects your local copy!
- **add** updated files to a "staging" area in preparation for committing and sharing, creates snapshot
- **commit** files in the staging area as a new version of your local copy – include comments on what was changed
- (pull again to make sure your changes don't conflict with others' changes – merge to resolve conflicts as needed)
- **push** commits from your copy to the central version (origin) to share with others
Using Android Studio git plugin

- access through terminal window
- can use integrated tools/menu
Katie's Lecture

- See her posted slides on Piazza also
Team Design Presentations – in pairs
3-4 teams per day (6-8 pairs)
students review designs during presentations

DAYS 13-14
3/15-17/2017
SPRING BREAK!!

3/22-24/2016
Team Design Presentations – in pairs
3-4 teams per day (6-8 pairs)
students review designs during presentations

DAYS 15-16
3/29-31/2017
Database Review
Data for Fragments

DAY 17 - 4/5/2017
Databases in Android

- SQLite is a slimmed down version of SQL, Structured Query Language, for performing operations on relational databases.
- Core Database class: SQLiteOpenHelper
- Database Adapter class: recommend to write one to make handling the SQL operations easier from the app activities.
- Example: YardPro1.zip
SQLite in a DBAdapter

- we define the attributes (columns) we want our records to have
- we create tables of records with specific columns of interest
- each time we define (or redefine) a table in a database, we need to change the database version number so it gets updated properly
- ContentValues is an object that holds the data for one record
- We use a Cursor (like an interator) to move through the records in a database table (or resulting from a query).
- see JobDBAdapter.java
Databases & ListViews

- typically the items in a listview correspond to records in a database
- there is an intermediate layer of a class object connecting the two (JobItem)
  - the toString may be used to populate what gets displayed about an object in the list item layout
- the class objects are collected into an ArrayList
- some form of ArrayAdapter is used to connect the ArrayList of objects to the ListView (JobItemAdapter)
- the position in the ListView is used to find the record in the database
Implementing common actions

- first find out which item is chosen based on the item ID
- get the corresponding record from the database
- make the changes to the record (see next slide)
- update the database
- update the listview and redisplay

- delete: just get item ID and delete corresponding record
Edit item actions

- edit more complex
  - may need to populate an item view with existing data
  - can use extras bundled into Intent to pass current data to an edit activity
  - may need to track whether adding item or edit item if the same detail activity is used for both
  - need to update individual fields in database

- if same activity used for edit & add an item – different save actions
  - check intent to see how it was called
  - update and finish (return) if in edit mode
Data & Fragments

- Fragment level: Bundle
- Activity level
  - getActivity()
  - SharedPreferences
- Application level
  - getActivity().getApplicationContext()

Remember that onCreateView() is the most important method for initializing Fragments!
Fragment/Database Exercise

Example: FragDbase.zip

TODO:
- download code from main website
- add delete all button to fragment1
- add code to delete all items from database that match the current item text
- add dialog to confirm
Content Providers
Preference Activities
Styles in Android

DAY 18 - 4/7/2017
Content Providers & Implicit Intents

- landscape (yardpro) example: camera, contacts
- We launch content providers by creating implicit intents
- Creating an implicit intent specifies a general action to be performed, but not a specific activity to handle it
- Any application component can be declared in the manifest as a filter (handler) for a particular intent type
- When an implicit intent is launched, the system finds an appropriate target and starts it
- We can "putExtra" to include string data in an intent, which is useful when a component from another application is handling it
Styles Overview

- create style resource file (mystyles.xml) in res/values
- can define multiple named styles in one resource file
- apply your style(s) to individual view elements, activities, or whole applications
- can inherit from parent styles – yours or built-in android ones
styles.xml Resource detail

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="MyFont"
    parent="@android:style/TextAppearance.Medium">
    <item name="android:textColor">#00FF00</item>
    <item name="android:typeface">monospace</item>
  </style>
</resources>
```

- items can be any View attribute
- only relevant ones will be applied to any element using the style
Applying Styles

- to a view in layout file:
  <TextView style="@style/MyFont" android:text="@string/hello" />

- to an activity in manifest:
  <activity android:theme="@style/MyFont"/>

- to an application in manifest:
  <application android:theme="@android:style/Theme.Translucent"/>
Style Inheritance

- from your own style (MyFont):
  
  ```xml
  <style name="MyFont.Red">
    <item name="android:textColor">#FF0000</item>
  </style>
  ```

- from Android styles:
  
  ```xml
  <style name="GreenText"
    parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
  </style>
  ```
Colors Overview

- Colors can be represented multiple ways:
  - as hexadecimal color code strings: #BFD9CD
  - as android color resources: @color/colorAccent

- Defining colors – best approach is in a colors.xml file in res/values:
  - we can override the default values for built-in resources here
  - we can also name our own custom colors

- See Katie's DriversEd for examples

- Web resource for hex color codes:
  - [https://www.w3schools.com/colors/colors_hexadecimal.asp](https://www.w3schools.com/colors/colors_hexadecimal.asp)
Style Limitations

- it's nearly impossible to dynamically apply styles in code
- some attributes require color resource, not literal value:

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme"
    parent="android:Theme.Light">
    <item
        name="android:windowBackground">@color/custom_theme_color</item>
</style>
```
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 fragment or activity
  - shared preference change listener
Preference Screen: Layout I

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

```xml
<?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 II

- 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:

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

- need class to extend PreferenceFragment (single set) or PreferenceActivity (multiple sets with headers)
- connect activity to xml definition in onCreate or onCreateView:

```java
public class MyPrefActivity extends PreferenceActivity {
    //...
    public void onCreate(Bundle myBundle) {
        super.onCreate(myBundle);
        addPreferencesFromResource(R.xmlprefs);
    }
    
    //...
}
```
Preference Activity Manifest

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

```xml
<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 the application level context
  
  ```java
  Context context = getApplicationContext();
  SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
  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
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
    }
Preferences Example

- settings in GPAcm
  - toggles A+ as 4.3 points, or 4.0 default
  - CourseList is SharedPreferencesListener
  - QuickCalc is not, but doesn't need to be since you can't get to the settings page from there

Mobile App Issues
Testing

DAY 19 – 4/12/2017
Mobile Issues & Best Practices

- Compatibility (BP for User Interface)
- Code efficiency (BP for Performance)
- Responsiveness (BP for Background Jobs)
- Seamlessness (BP for Interaction and Engagement)

See the Android -> Training -> Best Practices lessons for how to address some of these.
Compatibility issues

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

- 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
Code efficiency issues II

- 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 faster 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
More Seamlessness issues

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

- Black-box Testing: throw data at the program and see what happens
- White-box Testing: each possible path in a program (all possible decision cases and life cycle events) should be tested.
- Unit Testing: individually test each method with it's own mini driver method or program – incorporate white-box testing
- Regression Testing: when rewriting and updating code, be sure to re-test cases that worked before the "upgrade"
Testing – what to in general

- Boundary cases: the = part of <= or >=, first and last elements of lists or arrays, empty containers
- Valid values, action ordering
- Invalid values, unexpected actions
- Getting data
  - make up literal values
  - random values – helps with black-box testing
  - loops to generate data
Testing – what to in Android

- 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?
- various API devices/levels
- battery usage
- dependence on external resources
Testing – how to in Android

- 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 in shell for emulator

- monkeyrunner tool: an API and execution environment for test programs
Presentations

GOOD:
- expressiveness
- practicing
- moving through audience
- stay on topic with slides
- logical ordering to content
- motion to emphasize point
- speaking slowly
- start with overview

BAD:
- reading slides
- softspoken, drop pitch at end of sentence
- slides too cluttered
- inconsistent slide styles
- hard to read slides (color, fontsize, etc.)
- distracting motion
- no motion, hands in pockets
- 'aah', 'umm', 'like'
Demos

- different kind of presentation
- need to carefully plan out use cases for various tasks – know what you will input
- set up accounts ahead of time
- pre-populate with some data if needed
- timing is key
Firebase – networked database (Katie)
testing with monkey (Rachel)
Quiz 3
Quiz 3

- On Blackboard
- Login, will have 10 minutes to do
- Password is:
Front-end Blended Design Demos

DAYS 21-22
4/19-4/21
App Distribution

- Deployment preparation
- Cleaning code
- Publishing steps
Deployment Preparation

- clean-up code
- do aggressive testing
- set appropriate min-sdk, target-sdk, uses-features
- create documentation
Cleaning code

- eliminate unnecessary/unused components
- refactoring (use AndroidStudio tool for this)
- straighten out repositories

optimize app:
- aggressive testing to discover bugs
- simplify layouts to increase responsiveness
Packaging and release

- configure app for release
- build and sign release version APK (AndroidStudio Build menu or Gradle)
- release to test users
  - can distribute apk directly through email or website
  - alpha and beta testing for specific users in Google Play Developer Console -> APK
    - alpha and beta versions can't be rated/reviewed, so doesn't hurt final product rankings
Publishing Steps on Google Play

- Google Play
  - ratings and reviews for apps influence rankings
  - category selections enable targeted search
- set-up publisher account on the Google Play Developer Console
- set-up google wallet merchant account if selling anything
- use Developer Console to upload app versions, review usage stats, etc.
Societal Context

- **Challenges**
  - security & privacy
  - network connectivity
  - power consumption

- **Target Areas**
  - healthcare
  - developing countries

- **HCI/UI advances**
  - disabilities
  - interaction design
Quiz 4
Wrap-up
Team meetings
Wrap-up

- Course End Survey (link on Piazza)
- Phone/Tablet Return
  - by Monday 5/15!
  - required to get a course grade
  - return directly to Steve (CS support, Malone G61A)
Quiz 4

- On Blackboard
- Login, will have 10 minutes to do
- Password is:
Project Presentations

Reminder: final versions due by 2p on Saturday 5/13
(submit on Blackboard)

DAYS 25 & 26
5/3-5/5