CMU 04-801 C3: Advanced Mobile Application Development
Week 1: App Design Principles

With 2.9 million apps in the Google play store users have many choices when choosing an app.

- 25% of apps aren't used more than once
- 29% of users will switch to another app if an app doesn't meet their needs
    - 70% due to lagging load times
    - 67% too many steps
- 34% aren't opened more than 11 times

To attract and keep users, apps need to be well designed and provide a delightful app experience.

App Design

1. Design for the platform
    - Mobile, mobile, mobile
    - All apps should be easy to figure out and use
    - Make onboarding quick and easy
        - Download, install, start instantly, no lengthy startup
        - Avoid requiring initial signup/login
        - Show content immediately
    - Avoid unnecessary interruptions
    - Interaction is through taps and gestures
        - The comfortable minimum size of tappable UI elements is 44 x 44 pixels
    - Artwork and image should be useful and draw the user in
        - Adapt art to the screen size, high quality media is expected
    - Handle different device sizes and orientations
    - Leverage device capabilities

2. Content
    - Get users to the content they care about quickly
    - Provide only relevant, appropriate content that's useful to the immediate task
    - If in doubt, leave it out

3. Focus on the User
    - Apps should have a well-defined target user
        - target apps to a specific user level
    - Put the user in control
    - Think through the user flow
    - Get them to the relevant information quickly
    - Provide subtle but clear, immediate feedback
    - Create a compelling user experience
        - User interaction consistency

4. Focus on the task
    - Apps should have a well-defined goal
    - What problem is your app solving?
    - How is it better or different from other apps?

5. Respect the platform
    - Be familiar with the platform you're developing for

- Understand platform conventions and guidlines
- Users spend most of their time in apps other than yours and will expect similar interactions

Principles of Mobile App Design: Engage Users and Drive conversations
http://think.storage.googleapis.com/docs/principles-of-mobile-app-design-engage-users-and-drive-conversions.pdf (checklist at the end)

For each section what's a good or bad example of one of the principles in an app you use?

Chapter 1: App Navigation and Exploration

1. Show the value of your app upfront
2. Organize and label menu categories to be user friendly
    a. Be predictable
    b. Less is more
3. Allow users to go back easily in one step
    a. Back button to navigate in reverse-chronological order through the history of screens the user has recently worked with.
    b. Up navigation to navigate to the logical parent screen in the app's hierarchy.
4. Make it easy to manually change location
5. Create frictionless transitions between mobile apps and the mobile web.

Chapter 2: In-App Search

6. Prominently display the search field
7. Use effective search indexing
    a. spelling auto-corrections, recognition of root words, predictive text, and suggestions while the user enters text.
8. Provide filter and sort options

Chapter 3: Commerce and Conversions

9. Provide previous search and purchase information
10. Allow user reviews to be viewed and filtered
11. Enable comparison shopping features
12. Provide multiple third-party payment options
13. Make it easy to add and manage payment methods

Chapter 4: Registration

14. Provide clear utility before asking users to register
    a. Only ask a user to register if it's essential.
        i. apps with low brand recognition—or those in which the value proposition is unclear—must clear a higher hurdle when they ask users to register at the start of the experience.
    b. provide guest checkout at the point of conversion.
15. Differentiate sign-in from sign-up
    a. Use register instead of sign-up
16. Make password authentication a frictionless experience
    a. Use third party login
    b. Use fingerprint touch login

Chapter 5: Form Entry

17. Build user-friendly forms
    a. ensure that form fields are not obstructed from view by interface elements such as the keyboard.
    b. automatically advance each field up the screen.

c. include efficiencies like auto-populate, auto-capitalization, and credit card scanning.
d. Use the hint attribute instead of default text in an edittext so the user doesn't need to delete the default text

18. Communicate form errors in real time
19. Match keyboard with the required text inputs
    a. Specify keyboard type with the "inputType" attribute for EditText with values such as "phone" or "textPassword".
        i. Can combine values for various behavior
        ```
        android:inputType="textCapSentences|textAutoCorrect"
        ```
    b. Specify the user action button with the "imeOptions" attribute with an action value such as "actionSend" or "actionSearch".
20. Provide helpful information in context in forms

Chapter 6: Usability and Comprehension

21. Speak the same language as your users
    a. No big words
    b. Avoid jargon or unconventional terminology
    c. Be descriptive
    d. Be succinct
    e. Avoid truncation
    f. Make text legible
    g. Clear communication and functionality should always take precedence over promoting the brand message.
    h. Language will depend on your target user
        i. Expert vs novice will expect different language
    i. Mental model
        i. A mental model is a user's beliefs and understanding about a system based on their previous experiences
        ii. A users mental model impacts how they use a system and helps them make predictions about it
        iii. Mismatched mental models are common, especially with designs that try something new
22. Provide text labels and visual keys to clarify visual information
    a. Labeled icons are more easily understood
    b. Categories with visual indicators should include a key
23. Be responsive with visual feedback after significant actions
    a. Provide clear feedback
    a. Communication
        i. Toasts are used to provide the user with simple feedback in a small popup https://developer.android.com/guide/topics/ui/notifiers/toasts
            a. no action required from the user, not clickable
            b. fills a small amount of space
            c. automatically disappears
        ii. SnackBars are very similar to toasts but more customizable. Snackbars animate up from the bottom of the screen and can be swiped away from the user. They are prominent enough that the user can see it, but not so prominent that it prevents the user from working with your app. They automatically disappear like Toasts. https://developer.android.com/training/snackbar

        a. Snackbars allow you to add an action for a user response
            i. A button will be added to the right of the message
        b. Snackbars supercede Toasts and are the preferred method for displaying brief, transient messages to the user
    iii. A dialog is a small window that prompts the user to make a decision or enter additional information.
    https://developer.android.com/guide/topics/ui/dialogs
        a. Dialogs take over the screen and requires the user to take an action before they can proceed.
        b. They are disruptive as the user must act on them in order to continue.
        c. Only use a Dialog when the user's response is critical to your app flow as they are disruptive, otherwise use a SnackBar or Toast.

24. Let the user control the level of zoom
25. Ask for permissions in context
    a. Communicate the value the access will provide
    b. Users are more likely to grant permission if asked in the context of the relevant task
    c. Users are only prompted once so you want to ask them at the most likely place for them to agree
        i. Users can change permissions in settings but many won't
    b. Every Android app runs in a limited-access sandbox. If an app needs to use resources or information outside of its own sandbox, the app has to request the appropriate permission.
    c. You declare that your app needs a permission by listing the permission in the app manifest using a <uses-permission> element. In Android 6.0 Marshmallow(API 23) the permission system was redesigned so apps have to ask users for each permission needed at runtime. You must call the permission request dialog programmatically.
        i. When the user responds to your app's permission request, the system invokes the onRequestPermissionsResult() method, passing it the user response. Your app has to override that method to find out whether the permission was granted and behave accordingly.
        ii. If an app tries to call some function that requires a permission which user has not yet granted, the function will suddenly throw an exception and the application will crash.
        iii. Your app must handle if the user denies permission or selects the "Don't ask again" option in the permission request dialog
        iv. Also, users are able to revoke the granted permission anytime through a device's settings so you always need to check to see if they've granted permission and request again if needed.

Small group discussion:
In your apps from last semester how could you improve them in at least 3 of these areas?

Material Design
Android Design https://developer.android.com/design
Core App quality https://developer.android.com/docs/quality-guidelines/core-app-quality
Material Design https://www.material.io/

Google launched Material design in 2014 to provide guidance and advice to developers designing and developing apps. (not just Android)
Material is an adaptable system of guidelines, components, and tools that support the best practices of user interface design. Backed by open-source code, Material streamlines collaboration between designers and developers, and helps teams quickly build beautiful products.
Making Material Design
https://www.youtube.com/watch?v=rrT6v5sOwJg&list=PL8PWUWLnnIXPD3UjX931fFhn3_U5_2uZG

Design
Material System Introduction https://www.material.io/design/introduction/

- Material Design is a visual language that synthesizes the classic principles of good design with the innovation of technology and science.
- Design elements are not present just to be pretty, they are present to do something in a way that's logical and easy to use.

Material Foundation Overview https://www.material.io/design/foundation-overview/

- Layout
  - Predictable, consistent, responsive
  - Pixel density https://www.material.io/design/layout/pixel-density.html - pixel-density
- Color https://www.material.io/design/color/the-color-system.html
  - Material Design color tool https://www.material.io/tools/color/
  - Material Design Palette https://www.materialpalette.com/
- 2014 Material Design color palettes
- Typography https://www.material.io/design/typography/the-type-system.html
  - Type scale
  - Font size units
    - always use the "sp" (scale-independent pixel) unit for font sizes as it allows the font size to scale for the user based on their settings
- Iconography https://www.material.io/design/iconography/product-icons.html
  - Launcher icons represent your app. They appear on the home screen, settings, sharing, and can also be used to represent shortcuts into your app

Material Guidelines

- Material Theming https://www.material.io/design/material-theming/overview.html - material-theming
  - Themes are used for create and apply a design across your mock-ups and app
- Implementing your theme

Components
Details on designing and creating various components across platforms.
Android Components https://www.material.io/develop/android/components/dialog/

Develop
Material Design Develop https://www.material.io/develop/
Android Components https://www.material.io/develop/android/components/dialog/
Theming

- Color theming – default colors

- Dark theme
    - The DayNight themes are for the new Dark Mode that is part of Android 10. Before that it was up to each app individually to implement it, or the OEMs that added on dark/night mode support.
- Typography theming – default styles

Android styles and themes let you separate the app design from the UI structure and behavior, similar to CSS in web design.
Styles and themes are declared in the style resource file res/values/styles.xml.
A style is a collection of attributes that specify the appearance for a single View.

- You can only apply one style to a View, but some views like TextView let you specify additional styles through attributes such as textAppearance on TextViews.
- You can extend an existing style from the framework or support library or your own project using the "parent" attribute.
- They are not inherited by child views

A theme is a type of style that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply your style as a theme, every view in the app or activity applies each style attribute that it supports.

To apply a theme use the `android:theme` attribute on either the `<application>` tag or an `<activity>` tag in the `AndroidManifest.xml` file.

Android lets you set styles attributes at multiple levels (global to local, with local taking precedence):

- applied based on the global attribute (from current theme)
- applied based on the global style of the view (from current theme)
- applied based on the style of the view
- applied based on the style in the XML layout
- applied programmatically

You should use themes and styles as much as possible for consistency. If you've specified the same attributes in multiple places, Android's style hierarchy determines which attributes are ultimately applied.

AndroidX Library

Initially when Material Design was released there was no official support on how to implement it. Then Google released the Design support library to help developers implement Material Design. Last year Google released AndroidX which is the next evolution of the support library and implements Material Components for Android. With this release the support library won't be maintained in the future so we'll need to use AndroidX.

All new projects are automatically created using AndroidX. The app Gradle file will have the following dependency:

```
implementation 'androidx.appcompat:appcompat:1.1.0'
```

To use Material Components you need to do the following:
Open the Gradle file for your app (Module: app)

You need to have a compileSdkVersion of 28 or later:
```
compileSdkVersion 29
```

Add the library to the dependencies:
```
implementation 'com.google.android.material:material:1.1.0-rc01'
```

Now you can use the Material Components themes to get their styles and theme-level attributes.
In the Android manifest file you'll see the activity app theme
```
android:theme="@style/AppTheme"
```
If you open res/values/styles.xml you'll see where the style name AppTheme is defined.

When you create a project with Android Studio, it applies a theme to your app by default, as defined in your project's `styles.xml` file. This `AppTheme` style extends a theme from the support library.
```
<style name="AppTheme"
parent="android:Theme.AppCompat.Light.DarkActionBar"> </style>
```

Use these Material Component themes to get the latest component styles and theme-level attributes:

- Theme.MaterialComponents
- Theme.MaterialComponents.NoActionBar
- Theme.MaterialComponents.Light
- Theme.MaterialComponents.Light.NoActionBar
- Theme.MaterialComponents.Light.DarkActionBar
- Theme.MaterialComponents.DayNight
- Theme.MaterialComponents.DayNight.NoActionBar
- Theme.MaterialComponents.DayNight.DarkActionBar

Change the AppTheme style to use a material components theme as the parent.
```
parent="android:Theme.MaterialComponents.Light.DarkActionBar"
```

You can also define different styles for different Android versions if you want to use newer features while still being compatible with old versions. All you need is another `styles.xml` file saved in a `values` directory that includes the resource version qualifier.
res/values/styles.xml # themes for all versions
res/values-v21/styles.xml # themes for API level 21+ only
The common structure to do this is to define a base theme and then use it as the parent for any version specific themes so you don't need to duplicate any styles.

Resources
https://material.io/resources/
Tools

- Color tool https://www.material.io/resources/color/

Android also provides Quality Guidelines https://developer.android.com/develop/quality-guidelines/

CMU Welcome
Let's create a little sample app to play around with some app design. This will also act as a refresher for the basic setup of an app.
From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)
In the Phone and Tablet tab pick Empty activity.

Name: HelloAndroid
Package name: the fully qualified name for the project
Save location: the directory for your project (make sure there is a directory with the name of the project after the location)
Language: Java
Minimum SDK: API 21: Android 5.0 Lollipop (21 is the minimum API for Material Design)
Leave instant apps unchecked (we don't need to support instant apps)
Finish

(Preferences | Editor | Color Scheme to change scheme to presentation)

All new projects are automatically created using AndroidX.
To use Material Components open the Gradle file for your app (Module: app) and change compileSdkVersion to 29.

```
compileSdkVersion 29
```

Add the library to the dependencies:

```
implementation 'com.google.android.material:material:1.1.0-rc01'
```

The AndroidManifest file describes the fundamental characteristics of the app and defines each of its components. Every part of your app will be declared in this file. It acts as a blueprint for putting all the pieces of an app together.
Open the AndroidManifest file and look at what theme it's using. This theme is defined in the res/values/styles.xml file. The colors used are from the res/values/colors.xml file.
Let's create a new theme using a MaterialComponent theme as its parent. The parent themes provide a base level of design.
Using the Material Color tool https://www.material.io/resources/color pick out some colors and then Export for Android. Open the exported file and copy the colors into the colors.xml file. For this app you only really need the primary colors.

```
<color name="primaryColor">#c62828</color>
<color name="primaryLightColor">#f05545</color>
<color name="primaryDarkColor">#8e0000</color>
<color name="primaryTextColor">#ffffff</color>
```

In styles.xml create a new theme using these colors.

```
<style name="CMU" parent="Theme.MaterialComponents.DayNight">
    <item name="colorPrimary">@color/primaryColor</item>
    <item name="colorPrimaryDark">@color/primaryDarkColor</item>
    <item name="colorOnPrimary">@color/primaryTextColor</item>
</style>
```

We use @color because the style values are references to our color resources.
To apply this theme update the AndroidManifest.xml file. Themes can be applied to the whole application or just one activity. We'll apply it to the whole application.

```
android:theme="@style/CMU"
```

Go into the layout/activity_main.xml layout file.
Notice the textview has `android:text="Hello World!"`
It's not good practice to hard code string values in your xml file, you should use string resources. strings.xml in the values folder stores all the string resources.

Open strings.xml and you'll see there's already a string for the name of the app.
Add this resource with some initial value so you can see it worked.

```
<string name="heading">CMU</string>
```

Back in activity_main.xml change the textview to `android:text="@string/heading"`
The '@' sign means it's defined as a resource.
Now go into Design mode or Preview and you will see our textview has the new string value.
You can click anywhere on `@string/text_message` and click cmd-B (mac) (cntrl B on a pc) to automatically open the strings.xml file where the values for the string resources are stored.

Let's also make the text larger by setting the textAppearance attribute. You can use some of the predefined MaterialComponents values

```
android:textAppearance="@style/TextAppearance.MaterialComponents.Headline4"
```

Before we add more components, make sure Autoconnect is on. It should not have a line through it. This will automatically add constraints when new components are added to the layout

To let the user enter their name we need an EditText. But instead of adding one directly use TextInputLayout instead. This acts as a wrapper for a TextInputEditText to enable a floating label for hint animations.
Always add a hint (should be in the TextInputLayout) so the user knows what data is expected. Don't use the text attribute for this as the user will need to delete it before they can enter their value.
Remember to use a string resource for the hint.

TextInputEditText is a sub-class of EditText and is designed for use as a child of TextInputLayout. If you use an EditText here you'll get a warning telling you not to.
Assign an id to the TextInputEditText so we'll be able to get the value the user enters. (editTextName)

If the TextInputLayout has an error that some constraints are missing, add them. Every view must have at least two constraints: one horizontal and one vertical. Any view missing constraints at run time will be drawn at position [0,0] (the top-left corner) even though it's shown differently in the layout.

- A missing vertical constraint will cause it to be drawn at the top of the screen
- A missing horizontal constraint will cause it to be drawn at the left of the screen

Even though the app doesn't do anything yet, it's a good idea to run it (emulator is fine) to make sure it looks good and runs without errors.

Add a button, using a string resource for its text. Make sure it has an id (button)
Add a textView where we'll eventually write a message (leave the default text for now to make layout easier). Make sure it has an id (textView2)
Add an image to your project by adding it into the drawable folder. Make sure the file name uses letters, numbers, or underscore but no dashes.
Add an imageView to your layout and select the image you just added to your project. Make sure it has an id (imageView).
Use constraints to define how much room it should take up in your layout and then set layout_width and layout_height to match_constraint (which will change it 0dp). wrap_content adapts it to the size of the content which you don't want if your image is large.

Run it and see how the layout and theme colors look.

If the text in your toolbar is black but you were expecting it to be white, you'll need to specify that. Back in styles.xml add to your style

```
<style name="CMU" parent="Theme.MaterialComponents.DayNight">
    ...
    <item name="toolbarStyle">@style/CMU_Toolbar</item>
</style>

<style name="CMU_Toolbar" parent="Widget.MaterialComponents.Toolbar">
    <item name="titleTextColor">@color/primaryTextColor</item>
</style>
```

Here we're creating a new style called CMU_Toolbar that takes the Material Component Toolbar and customizes the color of the title text. We then use that in our CMU style as the toolbarStyle.

Now let's add the Java in MainActivity.java that will take the user's name and write a message to the second textView and set the image to the cmuscottydog image. Since we'll be doing this programmatically, remove the default text from the textView and remove the srcCompat value from the ImageView.

```
public class MainActivity extends AppCompatActivity {

    private EditText nameEditText;
    private TextView message;
    private ImageView image;
    private Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nameEditText = findViewById(R.id.editTextName);
        message = findViewById(R.id.textView2);
        image = findViewById(R.id.imageView);
        button = findViewById(R.id.button);

        //listener
        View.OnClickListener onClickListener = new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String name = nameEditText.getText().toString();
                message.setText("Welcome to CMU " + name + "!");
                image.setImageResource(R.drawable.cmuscottydog);
            }
        };

        //add listener to button
        button.setOnClickListener(onClickListener);
```

```
    }
}
```

Discussion points:
Why are my 4 variables defined at the class level?
Why are they private?
Why must the calls to findViewById be after setContentView?
What is the R class?
How would you test this app?

What if we want to make sure the user entered a name? Let's present a toast if they didn't.
This also clears the message textView and the ImageView in case they had already been set.

```
public void onClick(View v) {
    String name = nameEditText.getText().toString();
    if (name.isEmpty() == false) {
        message.setText("Welcome to CMU " + name + "!");
        image.setImageResource(R.drawable.cmuscottydog);
    } else {
        //source: https://stackoverflow.com/questions/6643432/remove-
the-image-from-a-imageview-android
        image.setImageDrawable(null);
        message.setText(" ");

        //toast
        Context context = getApplicationContext();
        CharSequence text = "Please enter your name";
        int duration = Toast.LENGTH_LONG;
        Toast toast = Toast.makeText(context, text, duration);
        toast.show();
    }
}
```

What happens when you rotate the device?
Because the activity is killed and then restarted, anything done programmatically is lost.
As your activity begins to stop, the system calls the onSaveInstanceState() method so your activity can save state information to an instance state bundle. The default implementation of this method saves transient information about the state of the activity's view hierarchy, such as the text in an EditText widget.
To save additional instance state information for your activity, you must override onSaveInstanceState() and add key-value pairs to the Bundle object that is saved in the event that your activity is destroyed unexpectedly. If you override onSaveInstanceState(), you must call the superclass implementation if you want the default implementation to save the state of the view hierarchy.
When your activity is recreated after it was previously destroyed, you can recover your saved instance state from the Bundle that the system passes to your activity. Both the onCreate() and onRestoreInstanceState() callback methods receive the same Bundle that contains the instance state information.
The system calls onRestoreInstanceState() after the onStart() method only if there is a saved state to restore. You can then retrieve any data you saved using the same keys.

So if the user has entered a name and clicked submit, when they rotate the device I want the message and the image to still be present. But you never want to duplicate code so I took the in the onClick method and moved it into a separate method called welcome(). This lets me call welcome() when restoring state and not duplicating the logic.

```
@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    if (name != null) {
        outState.putString("username", name);
    }
}

@Override
protected void onRestoreInstanceState(@NonNull Bundle
savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    name = savedInstanceState.getString("username");
    if (name != null && name.isEmpty() == false) {
        welcome();
    }
}
```

Launcher icons
https://developer.android.com/studio/write/image-asset-studio.html

- Android projects include default icons - ic_launcher.png and ic_launcher_round.png
- Launcher icons go into density specific res/mipmap folders (i.e. res/mipmap-mdpi)
- If you only include the higher resolution version of the icon Android will generate the lower resolutions
- There are also required icons for the action bar, dialog & tab, notifications, and small contextual
- Tablets and other large screen devices request a launcher icon that is one density size larger than the device's actual density, so you should provide your launcher icon at the highest density possible.

Add your own launcher icon to your app.
Select the res folder right-click and select New > Image Asset
Or File > New > Image Asset.

Icon type: Launcher Icons (Adaptive and Legacy)
Use Adaptive and Legacy if you're supporting Android 8, otherwise you can just do Legacy.
Name: leave as ic_launcher so you don't need to change it in the Android_manifest.xml file

Foreground Layer:
Layer Name: ic_launcher_foreground

- Select Image to specify the path for an image file. (pumpkin.png)
- Select Clip Art to specify an image from the material design icon set.
- Select Text to specify a text string and select a font.

Scaling – trim and resize as needed.

Background Layer:
Layer Name: ic_launcher_background

- Asset Type, and then specify the asset in the field underneath. You can either select a color or specify an image to use as the background layer.

Scaling – trim and resize as needed.

Next
Res Directory: main
Output Directories: main/res

Finish
Image Asset Studio adds the images to the mipmap folders for the different densities.

You can see the various launcher files created in the mipmap folder.

Now run your app and look at your launcher icon by clicking the right button, or go to the home screen.

You can download the Android Icon Templates Pack
https://developer.android.com/guide/practices/ui_guidelines/icon_design.html
You can also use Android Asset Studio to create all your launcher images.
http://romannurik.github.io/AndroidAssetStudio/index.html