Outline

- Logistics
- Fragments
- Services
- Android testing
- App architecture
- Mobile gaming
- Alternatives to native app development
Week 1

Industry panel discussion

Assn 1: Tip Calculator
Assn 2: Google Maps
Assn 3: Yelp Clone
Fragments

- A reusable portion of UI that lives inside an Activity.
- Multiple fragments can be combined in one activity
  - Helps to handle different devices and screen sizes
  - Helps to reuse common UI across your app
- Has its own lifecycle similar to the Activity lifecycle
Fragment vs Activity

- Methods defined on the activity are not available in the fragment
  - Need to use the activity property to access the enclosing activity
- Passing/accessing information in a fragment (intents/bundles) is done by asking the enclosing activity
- Fragment initialization and lifecycle are different
  - Activity: onCreate
  - Fragment: onActivityCreated
Activity lifecycle
Using fragments

- **Static fragment**
  - Add the `fragment` component in the activity

- **Dynamic fragment**
  - Add container to the layout and programmatically add the fragment
Code pointer

FragmentDemo

First fragment!!
Services

● A background task used by an app
  ○ Use to perform long-running operations without a UI
  ○ Examples: handle network transactions, play music, perform file I/O
● Also has a lifecycle
  ○ onCreate, onStartCommand, onDestroy
● Can broadcast a result when a task is completed
  ○ Applications can hear broadcasts using a BroadcastReceiver
Android testing

Why is it hard?

- **End to end testing**: directly testing the UI is flaky, e.g. dropdown takes time to render
  - Espresso: requires an emulator
- **Integration testing**: check interaction between different components
  - Robolectric: mock Android components
- **Unit testing**: just one component, no interaction with Android framework
  - JUnit/Mockito: very fast to run
App architecture

● Architecture: “fundamental structures of a software system”
● Architecture educates how you organize your code
  ○ Has consequences around how easy it is to debug your code or onboard new devs
● Different components of all mobile apps:
  ○ UI or view layer
  ○ Data classes or models
  ○ Repository: holds or retrieves the data
  ○ “Business logic” component for responding to user input
Android app architectures

- **Objective**: avoid having all logic live inside Activities

- **MVC**: Model View Controller
- **MVP**: Model View Presenter
- **MVVM**: Model View ViewModel
Gaming
Why use game engines?

- Cross platform capability
- Handles complexities around physics, lighting, 2D/3D graphics
- Games typically don’t have as many platform-specific differences
Alternatives to native app development

What problem are they trying to solve?

1. Double the effort to ship an app two platforms (Android + iOS)
2. Developer experience sucks- gatekeepers to releasing your code
React Native

- Developed by Facebook for cross platform app development (move faster)
- Uses JavaScript, source code is converted to native elements
- Tight platform integration, ability to write modules in platform code
- Heavily used across industry: Facebook, Uber, Walmart
Flutter

- Created by Google
- UI toolkit for building native apps for mobile, web, and desktop in a single codebase
- Provides a full native experience by using native compilers
- Uses the Dart programming language
- Less mature than React Native but growing
Xamarin

- Bought by Microsoft
- Uses .NET and C# (Microsoft technologies)
- Also turns into native code
- 75% code sharing between platforms
- Not as popular in Silicon Valley, but has a mature community
If you want to publish your app...

- Create a [Google developer account](#)
- Create assets for publishing: icon, screenshots, description
- Use a unique package name
- Create a keystore in Android Studio and signing key for the APK
- (Optional) Use Proguard to minify your app
- (Optional) Add an analytics/crash tracking library
- Publish + profit!