ANDROID IX

UI Part 1
- **Views**
  - Rectangles that appear on the screen
  - All widgets, UI elements inherit from View
  - Handles drawing, event-handling
  - Examples: TextView, Button, Radio Buttons, Checkboxes, Date Pickers, . . .

- **ViewGroup**
  - a container for children views
  - Needed to display multiple views
  - Layouts
Checkboxes and Radio Buttons

- Checkboxes: define in xml and in your onclick method, check if box is checked
- Radio Buttons: define RadioGroup viewgroup, with RadioButton children, and in your onclick method, check to see which button was checked
Layouts

- Frame layout: one element in each z dimension
  - multiple elements are stacked z-wise
- Linear layout: lays elements out in a line: you’ve seen this before
  - Can specify weights
Relative Layout

- Allows elements to be positioned relative to each other
- It’s being replaced by the more powerful Constraint Layout
View Width/Height

- width = wrap_content, height = wrap_content
  - It's a view party

- width = match_parent, height = wrap_content
  - It's a view party

- width = wrap_content, height = match_parent
  - It's a view party

- width = match_parent, height = match_parent
  - It's a view party
gravity: contents within element

layout_gravity: relative to parent
Padding vs Margin

padding = x

layout_margin = x
View Visibility

visible

Rain

invisible

Rain

gone

Rain
Quiz

Which will the layout render, 1, 2, or 3?
**Constraint Layout**

- Allows positioning of elements relative to each other by using constraints
  - Similar to RelativeLayout, but avoids nesting
  - Copied off of iOS???!!

  ![Image](apple.png) Say it isn’t so!!

- You must add at least one horizontal and one vertical constraint for the view to another view
Constraint Layout Toy App: Boarding Pass

Let’s look at the app!
Avoid excessive nesting

- Suggested Max:
- 80 views
- 10 nested viewgroups
- Which is more efficient, A or B?
**Data Binding**

- Values of UI elements can be changed at runtime with `findViewById`
  - But this is computationally expensive
- Another way: Data binding library
- Enable it in your `gradle.build` file

```java
buildTypes {
    release {
        minifyEnabled false
    }
}

// TODO (1) Enable Data Binding in your application
dataBinding.enabled = true;
```

- Make a POJO for the info you want to bind
- Make sure the root element of the layout to be bound is a layout
Data Binding Continued

● Create a data binding instance (ActivityMainBinding: autogenerated based on layout name)
● Get a POJO with real data (you’re responsible for that)
● Write a method (or something) that takes all the info in the POJO and sets the views in the data binding instance with this info
  ○ You still have to put all the data in yourself, but:
  ○ You don’t have to call findViewById, as the views will already be attributes of the data binding instance

```java
private void displayBoardingPassInfo(BoardingPassInfo info) {
    mBinding.textViewPassengerName.setText(info.passengerName);
    mBinding.textViewOriginAirport.setText(info.originCode);
    mBinding.textViewFlightCode.setText(info.flightCode);
    mBinding.textViewDestinationAirport.setText(info.destCode);
}
```
Landscape Mode?

- Constraint Layout will handle some layout changes gracefully.
- But sometimes it’s just better to make a separate layout for landscape mode.
- Do this by creating a new folder, layout-land, and put it in your res folder.
- Put a layout of the same name in this folder and adjust.
Including layouts

- Sometimes multiple layouts will share complex components
- The best way to handle these is make the components in a separate xml, and include them when needed

```xml
<include
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textViewPassengerName"
    android:layout_marginTop="16dp"
    android:id="@+id/flight_info"
    layout="@layout/flight_info"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintRight_toRightOf="parent" />
```