ATLS 4120/5120: Mobile Application Development
Week 2: Adaptive Layout

Adaptive Layout

https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout/
The goal of adaptive layouts is for your user interface to look good and work well on all iOS devices by adapting to the user's environment.

- Device size and settings
- Orientation
- Locale

Interface Builder gives us the tools to do this using auto layout, constraints, and size classes.

Orientation

- iOS devices can be used in portrait or landscape mode.
- There are four defined iOS interface orientations values for UIInterfaceOrientation https://developer.apple.com/documentation/uikit/uiinterfaceorientation
- iPhone
  - Most apps support autorotation but not all
    - Movies can be watched in landscape only
    - Contacts can only be edited in portrait mode
    - Immersive games often support only 1 orientation
  - Upside down portrait is not usually supported
- iPad
  - Recommended that all apps support every orientation
- For iPhone apps, the base rule is that if autorotation enhances the user experience, you should add it to your application. For iPad apps, the rule is you should add autorotation unless you have a compelling reason not to.
- Interface orientation can be set for the whole application and that will be the default for all view controllers
- You can also set valid orientations for each view controller.
- When the user rotates the device the active view controller calls **supportedInterfaceOrientations()** to see if the new orientation is supported. If it is then the app's window and view will be rotated and resized to fit the new orientation.
- Xcode
  - Target | General | Deployment info

Auto Layout

Auto Layout enables you to create a user interface that responds to different screen sizes, orientations, and localization.
Auto layout was first introduced in iOS6 and Xcode 4 and improved in subsequent versions
Auto layout also helps with internationalization as it can resize your labels based on the content

Intrinsic Content Size

All views have an intrinsic content size, which refers to the amount of space the view needs for its content to appear in an ideal state. For example, the intrinsic content size of a UILabel will be the size of the text it contains using whatever font you have configured it to use.

Constraints

Anatomy of a Constraint (image/slide)

- Constraints express rules for the relationship between UI elements in your interface
- These rules are used to automatically create your user interface when your app runs
- For each element there must be enough constraints to determine its placement
  - horizontal position
  - vertical position
  - height
  - width
- Auto Layout will calculate the width and height for views with a natural width and height without us forcing one by creating width and height constraints. This allows these views to grow and shrink based on the content in them
  - Labels, buttons
- Although Auto Layout does its best to give views the space they need based on their intrinsic content sizes, all views also have a content compression resistance priority and a content hugging priority that determine how much it fights to retain its intrinsic content size when available space is less than or greater than it needs, respectively.
- Creating constraints
  - Auto resizing
  - IB can add suggested or missing constraints based on your interface layout
  - Create and modify your own constraints

Auto Resizing

In the size inspector tab you can set auto resizing masks for simple layouts.
These masks are translated into constraints at run time.
If you use auto resizing on a view then you can't manually create constraints.
Once you create constraints manually, they override the auto resizing masks on that view.
You can mix and match on different views, but only use one type per view.

helloworld

(helloworld autolayout1)
In Main.storyboard use the editor options to preview your layout on other devices (4s or SE and Pro Max for the range of phone sizes). What looked centered for the iPhone11 isn't centered in these sizes. Go into the size inspector and use autoresizing to center the label (put text in temporarily so you can see it in Preview) and button.
You can center it horizontally by removing the leading and trailing pins and adding the horizontal arrow inside the box. Leave the top pin if you want that distance from the top.
For the button removing the leading and trailing pins and add the vertical arrow inside the box. Leave the top pin for the top spacing.
Should look good in portrait but if you rotate you'll see it only looks good in landscape if both the label and button are in the top half of the view, the bottom half is cut off in landscape orientation.
This method only works for super simple layouts (often storyboards).

Creating Constraints

Working with Constraints in Interface Builder
Let Xcode add suggested missing constraints
Manually create constraints

- Constraint properties
  - Attributes: leading and trailing are the default

  - Values: size or offset of constraint in points
  - Relation: type of relationship between elements
  - Priority: required, high, or low
- Align: align edges or centers of elements to each other or within the container
- Pin: sets width or height spacing for an element or to a view
  - Leading space (from left for English)
  - Trailing space (from right for English)
  - Top
  - Bottom
- Constraints are color coded
  - Blue: constraints are valid
  - Orange: constraints are misplaced or ambiguous
    - Dotted orange box shows constraints with mismatched views
    - Need to update frames or update constraints
  - Red: constraints are conflicting
- Layout issues are listed in the document outline and provide suggested fixes
- You can clear, reset, change, or add missing constraints to resolve issues
- Once you think you have the right constraints, Update Frames to see what it will look like for different class sizes using Preview

Stack Views

Stack views make it easy to layout objects in a horizontal or vertical row.
In Interface Builder you can drag from the object library a horizontal or vertical stack view onto your view and then add objects into it.
Or select objects in your view and embed in a stack view.
In the attributes inspector you can change the stack view attributes:

- Axis – vertical or horizontal orientation for the stack
- Alignment – how the elements in the stack are positioned
  - Fill – each element fills the size of the stack view
  - Leading – each element's leading edge is aligned to the leading edge of the stack view
  - Center – center of each element is aligned to the center of the stack view
  - Trailing - the trailing edge of each element is aligned to the trailing edge of the stack view
- Distribution - defines how the elements are distributed in the stack view (along specified axis)
  - Fill – elements are resized to fill all available space
  - Fill equally - elements are resized equally to fill all available space
  - Fill proportionally - elements are resized proportionally to fill all available space
  - Equal spacing – elements are positioned at an equal distance from one another
  - Equal centering – the center of each element is set to an equal distance to each other
- Spacing - defines the space between each element in the stack view.

helloworld

(helloworld autolayout2)
If your label is empty, put some text in it just for layout purposes.
Since we only have 2 objects and we want them in a column let's try a stack view.
Select the label and button (shift click or click and draw around them) and embed in a stack view (Editor | Embed in | Stack View or click the Stack icon on the bottom right, right most button).
(You can always undo this by selecting the stack view and selecting Editor | Unembed)
Note that in the document hierarchy the label and button are now embedded in the stack view.

With the stack view selected go into the attributes inspector and make sure the axis is vertical.
If needed, add some spacing so the label and button aren't right on top of each other.
If the stack view is not centered move it so it's centered horizontally and vertically.
Select the Stack View and use the Align button to select Horizontally in Container for the x position and Vertically in Container. Click Add 2 Constraints.
This should move the stack view to be centered and resolve any auto layout issues.
A non-zero constant value for an align center constraint is its offset from the center.
Clear constraints so we can use auto layout when it's not all centered.
(or keep the constraint to Align Center X and in the size inspector delete the Align Center to Y constraint)
Select the Stack View and use the Align button to select Horizontally in Container for the x position if you cleared all constraints.
Next to your View Controller Scene you'll see a red arrow. Click on the red arrow and you'll see there are constraint errors that need to be fixed.
You must fix these in order to have the correct layout at run time.
It's telling you that the stack view needs a constraint for the Y position.
Select the Stack View and chose Add New Constraint. To create a top constraint click the line to the top margin and Add one constraint.
The red arrow should now be gone.
Look in the size inspector to see the constraint added. Align Top to Safe Area aligns the top of the stack view to the top of the view's safe area. Click edit on that constraint to see that you can change its value. Double click on the constraint to see more info for the constraint. Change the constant to see the stack view move.
Now you have constraint rules and define the horizontal and vertical positioning of the stack view so there shouldn't be any errors and your layout should look good in portrait and landscape in different device sizes.
Move the stack view and the blue lines will be replaced with orange lines and you have an auto layout warning that the stack view is misplaced. This is because your layout(frame) doesn't match your constraints. If you run it you'll see that it will still use the constraints for layout.
To resolve the auto layout issues you can either:
Update frames – this will change the storyboard layout to match your constraints (constraints win)
Update constraint constants – this will update your constraint values to match your storyboard (storyboard wins)

Size classes
https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout/
Instead of thinking in terms of different device sizes, Apple generalized these into size classes.
Each iOS device has a default size class dependent on orientation and screen real estate (think split screen)
Size classes define height and width dimensions as either compact or regular and these two dimensions for a trait collection.

- Size classes enable a layout to work with all iOS device screen sizes and orientations
- Build your interface as it will look on most devices and orientations
- Change the interface for different size classes by varying traits
    - Size or position of a view
    - Add or uninstall a view or constraint
- When you vary for traits the variations are only for that size class

daVinci

(daVinci autolayout)
Let's try to get our da Vinci app to look good on different size screens.
Move the UI controls to where you'd like them.
Select the View Controller and in Resolve Auto Layout Issues in the section All Views in View Controller chose Add Missing Constraints.
This works for simple apps but if it doesn't work go back into the same menu and chose Clear Constraints.
Let's look at how we can create constraints manually, working on one UI control at a time, starting from the top.

Label
X position: align horizontally in container (align button or Align | Horizontally in Container)
As soon as you add 1 constraint Xcode realizes you're manually creating constraints. So now you'll get an error that there's no vertical constraint.
Y position: Pin top space to top margin(16)
Don't need to specify width or height as the label's intrinsic size will be used.
Many of the views that UIKit provides, including UILabel, are capable of having Auto Layout set their size based on their actual content. They do this by calculating their natural or intrinsic content size. At its intrinsic size, the label is just wide enough and tall enough to completely surround the text that it contains. When we run the application and click one of the buttons, the label's text will be set and its intrinsic content size will change. Try it.

Using Dynamic Font will automatically adjust the font. In the attributes inspector check the box and for font chose a text style (such as title 1) instead of system.
Use Preview to see how the label looks on different size devices.

When your constraints and the storyboard frame don't match you'll see dotted orange lines. You have to decide if you want to update the frame to match the constraint (Update | Selected Views | Update frames) or update the constraints to match the frame (Update | Selected Views | Update constraints).

Buttons
It makes sense to always have these buttons aligned horizontally so select both buttons and embed them in a stack view (Editor | Embed in | Stack View or click the Embed In icon on the bottom right, right most button)
With the Stack view selected go into the attributes inspector and change the distribution to Equal Spacing so the buttons are distributed equally (try the different options).
Spacing will change the amount of space between the buttons in the stack view.
Now we need to position the stack view so make sure it's selected the whole time now using the document outline.
X position: align horizontally in container (align button or Align | Horizontally in Container)
As soon as you add 1 constraint Xcode realizes you're manually creating constraints. So now you'll get an error that there's no vertical constraint.
Y position: Control click and drag from the stack view(use document outline) to the label. Add vertical spacing – top space constraint. Go into the size inspector and change the value(25). Stack view.top equal label.bottom.
Now you have all the constraints you should need. If you have a red arrow at the top of your document outline click on it to see what your missing.
If you see orange lines in your view select the stack view and Update Frames
Now it will show you where the buttons will be based on the constraints.

Use Preview to see how the buttons look on different size devices.

Image
X position: align horizontally in container (align button or Align | Horizontally in Container)
Y position: Control click on the image and drag to the stack view to add a vertical top space constraint (25)
I want to make sure the image's aspect ratio never changes.
Add New Constraint | Aspect Ratio (220:299)
(all above are required)
You shouldn't have any auto layout errors at this point but if you look in preview the image isn't scaling larger when there's more room.
If I pin the width and height to the size of the image then it will be cut off on the iPhone.
Pin width = 440
Pin height = 598
I really want these to be this width and height or smaller, so go into the size inspector and change both of these constraints to "Less Than Or Equal".
I don't want them larger because that's the size of the image and scaling up would look bad. The best thing to do is find a high resolution image and then scale down.
Now to make sure the image is as big as space allows I want it to scale up to the superview width if it can.
Control click from the image and drag to the right and chose Equal Widths.
Go into the size inspector and edit this constraint to High Priority(750). We still want width <=440 to be required. Change the constant to -20 if you want a border on the right and left.
Then I pinned the image to the bottom superview >= 10 so that it never went past the bottom margin.

Landscape
In Preview if the buttons disappear on a smaller iPhone in landscape it's because it's running out of room.
You can play around with different values to see if you can make it work.
Next week we'll look at how to design different layouts for different size classes.