iOS Dev Accelerator Week1 Day4

- Lazy Loading
- TableView HeaderView
- Nib's/Xib's
- Automatic Table View Row Height
- Technical Thursday: Stack Data Structure

The Struggle is Real

Homework Review

Lazy Loading

Lazy Loading

- “Lazy Loading is a design pattern commonly used in programming to defer initialization of an object until the point at which it is needed”
- In iOS, lazy loading is often time used in collection views and table views for deferring the loading of resources used by cells until the time those resources are actually needed.
- Swift has introduced a lazy keyword that can make properties lazy, so they wont be instantiated until they are accessed. (more on this later in the course)

Demo

`TableViewHeaderView`

The tableView’s header view

- In addition to header views for each section, you can also have an accessory view that sits onto of the table itself.
- It’s just a regular UIView
- Can be set in code by creating the view and setting it to the tableView’s tableHeaderView property.
- Set in storyboard by just dragging a view on to the tableview.

Demo

Nib’s/Xib’s

Nib’s/Xib’s

- Nibs, or Xibs since Xcode 3.1, allowed developers to create interfaces graphically instead of programatically.
- Storyboard are now the best (and Apple recommended) way to create your interface, but nibs still serve a few purposes:
    - Create a single isolated view controller layout
    - Create the layout of a non-view controller related view (like a tableview cell!)
    - Work better with source control

How Nib’s work

- Interface objects are what you add to a nib file to layout your interface.
- At runtime, the interface objects are instantiated by the system and inserted into your code.

Nib Lifecycle

1. The system loads contents of the nib file into memory
2. the nib's object graph is unarchived, with our old friend NSKeyedUnarchiver
3. All connections are reestablished (outlets and actions)
4. awakeFromNib is sent to all appropriate objects

When to use a Nib?

1. Use a Nib to layout a View Controller's interface in insolation (ie not in a storyboard)
2. Use a Nib to layout a Table View or Collection View cell in isolation (maybe because it will be used in multiple view controllers)
3. Use a Nib to layout a custom view that isn't created with a view controller

View Controller and Nib

1. Set your Nib's File's Owner as the View Controller's class (this allows you create the outlets and actions!)
2. Use your view controllers constructor that takes in a nib name and bundle (this is inherited from UIViewController)

Custom Views and Nib

1. Drag out a view into your Nib
2. set its class to be your custom views class (this allows outlets!)
3. Use the NSBundle method loadNibNamed:owner:options: to load th nib file. This will return an array of all the root objects of the nib
4. Grab the appropriate object from the array (it will be the only object if your nib only holds one view)

Cells and Nibs

1. Drag a table view or collection view cell into your Nib
2. Set the class of the cell to be your custom class (this allows you to drag outlets)
3. In viewDidLoad(), or some other appropriate setup method, call registerNib:ForCellReuseIdentifier: on your table view

Demo

Automatic TableView Row Height

Row Height

- Right now our table view gets it row height straight from the storyboard.
- Theres a datasource method we could implement, that requests the row height for every row that is displayed
- Prior to iOS 8, having dynamic cell height was a bit of a pain. You had to calculate the size of your text labels based on the font type, font size, and how many characters you had. You would do this in the data source method mentioned in the previous bullet

New way

- With iOS8, it is much easier:
  - Make sure you are using auto layout in your cell
  - Ensure the elements that are going to be dynamic don't have fixed height constraints
  - Modify the settings of the elements (for label set lines to 0, for text view disable scrolling)
  - Give your table view an estimatedRowHeight and then set its rowHeight to UITableViewAutomatic Dimensions

Technical Thursdays: Stacks

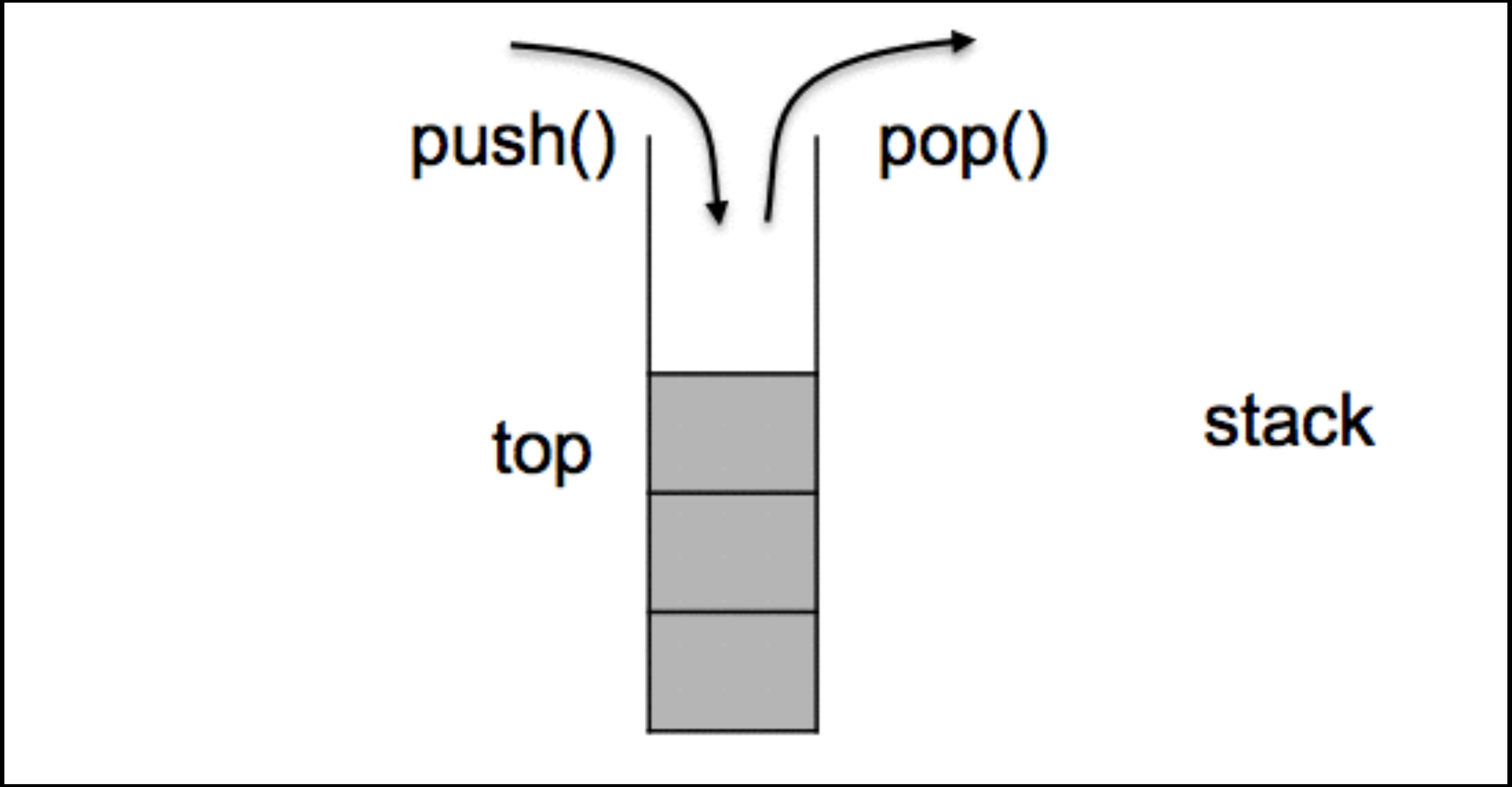
Stack Data structure

- “The stack data structure is one of the most important data structures in computer science” -Wikipedia (so you know its legit)
- Used in navigation controllers!
- And also in pretty much every programming language ever.

Stack Metaphor

- To understand a stack, think of a deck of playing cards that is face down.
- We can easily access the card that is on top.
- There are 2 things we can do to access the card on top:
  - peek at it, but leave it on the deck
  - pop it off the deck. When we pop something off a stack, we are taking it off the stack
- When you want to put something onto the stack, we call it pushing onto the stack
- A stack is considered LIFO. Last in, first out. This means the last thing we added to the deck (pushed) is the first thing that gets taken off (popped)

Stack Visual



Call stack

- “In computer science, a call stack is a stack data structure that stores information about the active subroutines of a computer program.”
- In most high level programming languages, the implementation of the call stack is abstracted away for us, but its still valuable to know what it is and how it works.
- When a function/method is called, it is pushed onto the stack. Any local variables are created and stored on the stack as well. If the called function calls another function, that 2nd function is pushed onto the stack as well. This keeps happening until all the functions have returned and they are all popped off the stack.

call stack demo

Implementing a stack

- There are two easy ways to implement a stack: using an array and using a linked list.
- Since we haven't learned linked lists yet, we will focus on the array way for now.

stack implementation demo