CSC 581: Mobile App Development

Spring 2019

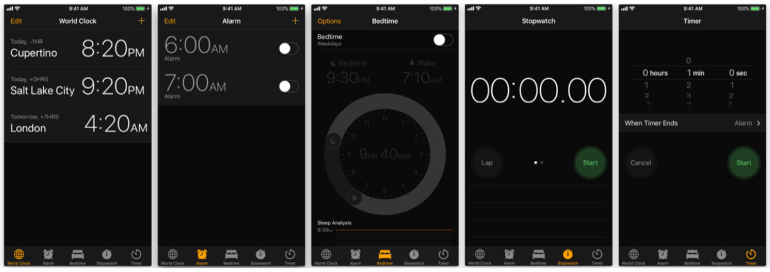
Unit 4: navigation & data persistence

- navigation design
  - tab bar view, design guidelines
  - viewDidLoad and other event-based methods
- reading text from a file
  - String.split, String.components
- reading/writing data
  - Info.plist, UserDefaults
- other (potentially useful) features
  - picker view, scroll view, table view
  - protocols, reading/writing objects

Tab bar view

a tab bar allows you to arrange your app into distinct sections

- e.g., Apple Clock app: World Clock, Alarm, Bedtime, Stopwatch, Timer



a tab bar is defined using a Tab Bar View

- shows tabs (icons) across the bottom of the screen
- each tab has its own navigation hierarchy
- the tab bar controller coordinates the navigation

Adding a tab bar

first, create the main View for your project

- select that View, then choose Editor > Embed in > Tab Bar Controller
- this adds a Tab Bar Controller to the canvas as well as adding a UITabBarItem to the bottom of the View

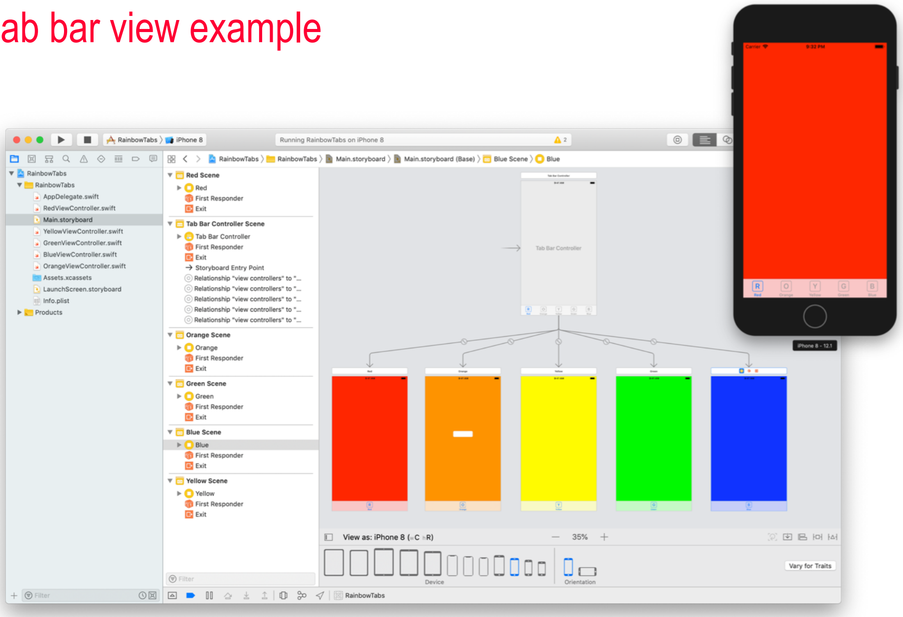
to add another tab bar item

- drag a View Controller from the Object Library to the canvas
- connect it by control-dragging from the Tab Bar Controller to the new View Controller, and choosing "view controllers" under Relationship Segue
- this adds a second tab bar item to the Tab Bar Controller
- if desired, create a ViewController class to define the behavior of the new view

customize the tab bar item

- select the desired tab bar item in a View
- in the Attributes Inspector, select the desired icon under System Item
  e.g., Favorites, Features, Search, Downloads, …
- you can change the title under Bar Item > Title
- you can select a custom icon image under Bar Item > Image

Tab bar view example



Navigation design guidelines

Design an information structure that makes it fast and easy to get to content.

- Organize your information in a way that requires a minimum number of taps, swipes, and screens.

Use standard navigation components.

- Whenever possible, use standard navigation controls, such as tab bars, segmented controls, table views, collection views, and split views. Users are already familiar with these controls and will intuitively know how to get around in your app.

Use a navigation bar to traverse a hierarchy of data.

- The navigation bar's title can display the user's current position in the hierarchy, and the Back button makes it easy to return to the previous position.

Use a tab bar to present peer categories of content or functionality.

- A tab bar lets people quickly and easily switch between categories or modes of operation, regardless of their current location.

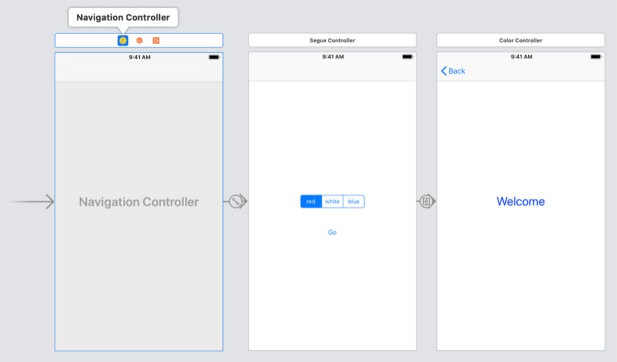
View event management

recall that a ViewController has a default viewDidLoad

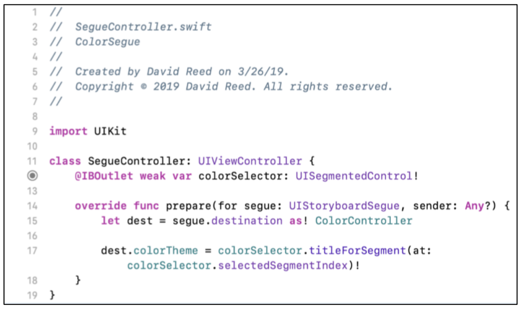
- this method is automatically called after the view loads
- useful if you need to set properties of the view, especially after a segue

e.g., suppose we wanted to select a color theme on the welcome screen

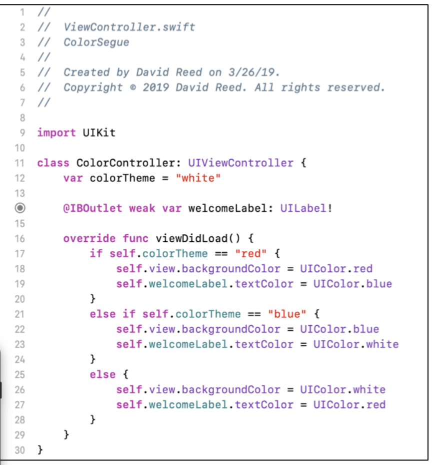
- then segue to a new screen that utilizes the selected color theme



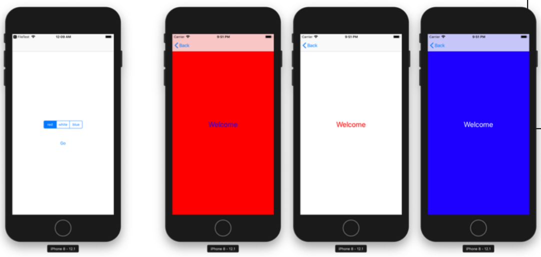
Color theme segue



```
//
//  SegueController.swift
//  ColorSegue
//
//  Created by David Reed on 3/26/19.
//  Copyright © 2019 David Reed. All rights reserved.
//

import UIKit

class SegueController: UIViewController {
    @IBOutlet weak var colorSelector: UISegmentedControl!

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        let dest = segue.destination as! ColorController

        dest.colorTheme = colorSelector.titleForSegment(at:
            colorSelector.selectedSegmentIndex)!
    }
}
```



```
//
//  ViewController.swift
//  ColorSegue
//
//  Created by David Reed on 3/26/19.
//  Copyright © 2019 David Reed. All rights reserved.
//

import UIKit

class ColorController: UIViewController {
    var colorTheme = "white"

    @IBOutlet weak var welcomeLabel: UILabel!

    override func viewDidLoad() {
        if self.colorTheme == "red" {
            self.view.backgroundColor = UIColor.red
            self.welcomeLabel.textColor = UIColor.blue
        }
        else if self.colorTheme == "blue" {
            self.view.backgroundColor = UIColor.blue
            self.welcomeLabel.textColor = UIColor.white
        }
        else {
            self.view.backgroundColor = UIColor.white
            self.welcomeLabel.textColor = UIColor.red
        }
    }
}
```



Other view event methods

in addition to viewDidLoad, there are other event-based methods

- viewWillAppear()
  performs tasks that need to be done each time the view is to appear on the screen
  e.g, refreshing views, adjusting to new orientation, accessing location

- viewDidAppear()
  waits until the view is fully loaded – better for complex or slow tasks
  e.g., starting an animation, fetching data

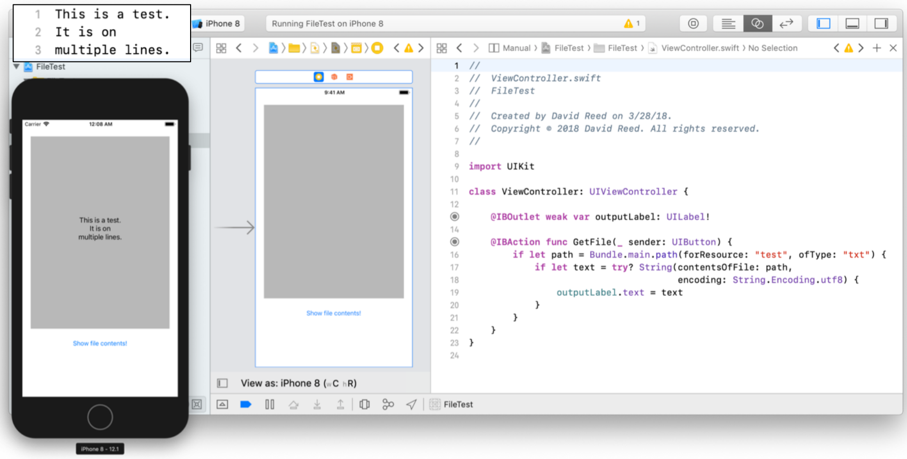
- viewWillDisappear()
  performs tasks that need to be done when the user navigates away from the screen
  e.g, refreshing views, adjusting to new orientation, accessing location

- viewDidDisappear()
  waits until the user has navigated to a new view
  e.g., stop services related to the old view (such as background audio)

Reading text from a file

it is fairly straightforward to add a data file to a Project & read from it

- add the file to the project Bundle by dragging into the Project Navigator
- utilize Bundle.main.path to specify the file name (here, test.txt)
- call String method to get the file contents (note use of try? since may not find it)



```
This is a test.
It is on
multiple lines.
```

```swift
//
//  ViewController.swift
//  FileTest
//
//  Created by David Reed on 3/28/18.
//  Copyright © 2018 David Reed. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var outputLabel: UILabel!

    @IBAction func GetFile(_ sender: UIButton) {
        if let path = Bundle.main.path(forResource: "test", ofType: "txt") {
            if let text = try? String(contentsOfFile: path,
                                      encoding: String.Encoding.utf8) {
                outputLabel.text = text
            }
        }
    }
}
```

Extracting content from text

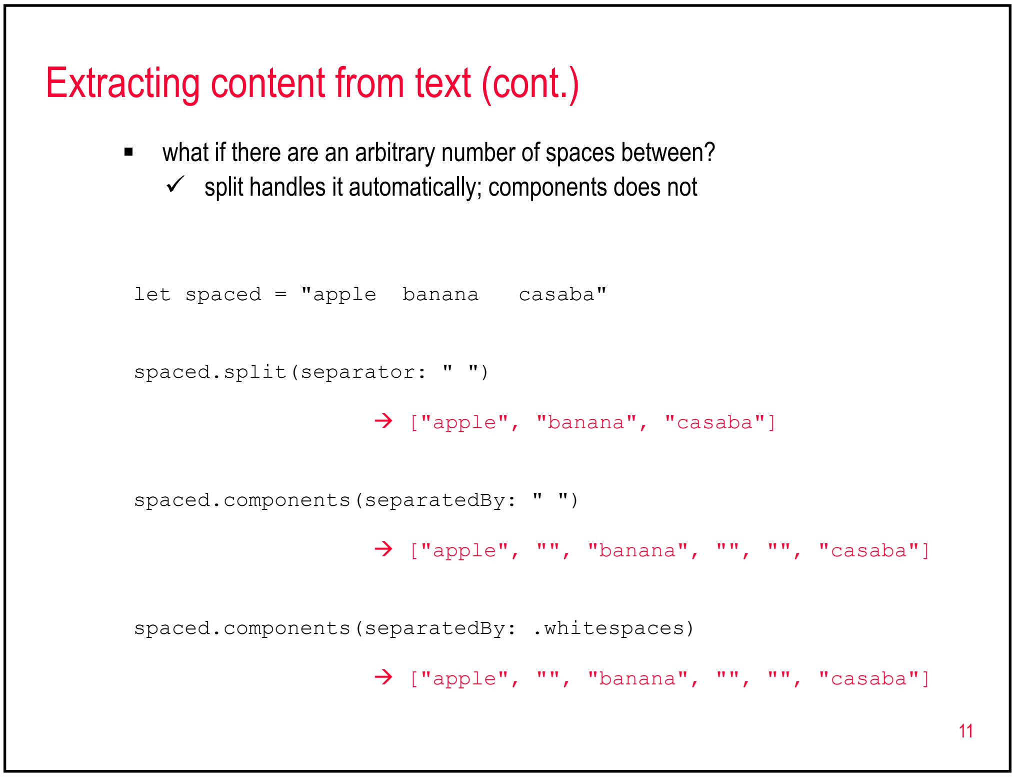
- to split a String into components based on a separator

```
String.split(separator: Character)
```

or

```
String.components(separatedBy: CharacterSet)
```

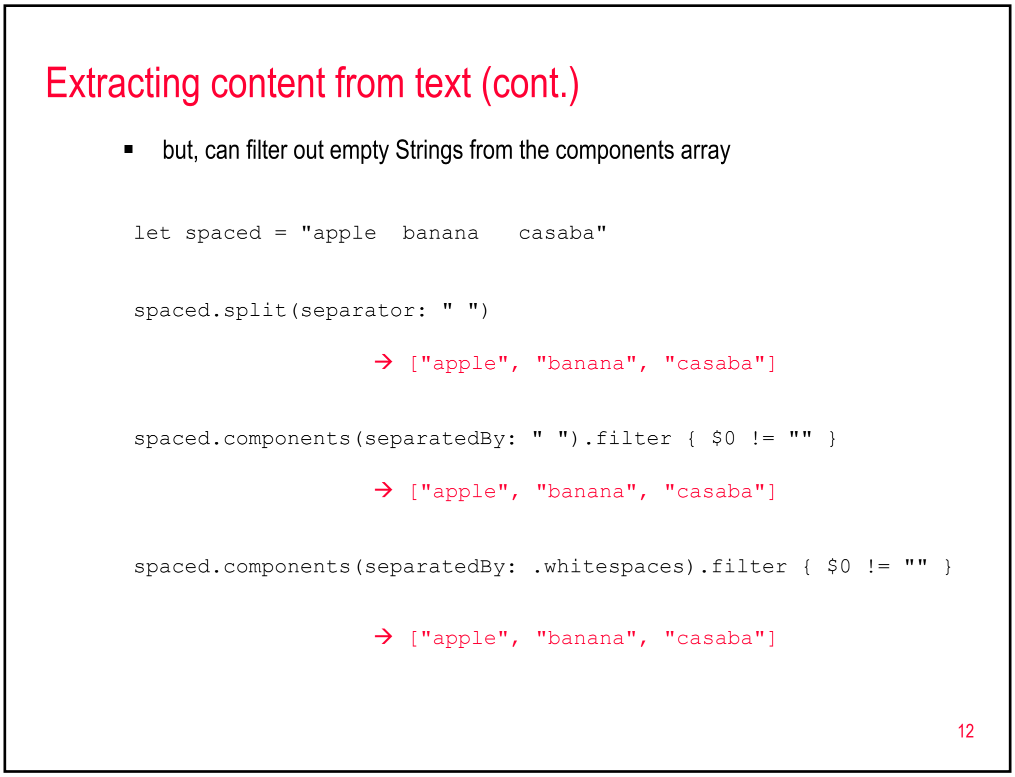
```
let text = "apple banana casaba"

text.split(separator: " ")

                         → ["apple", "banana", "casaba"]

text.components(separatedBy: " ")

                         → ["apple", "banana", "casaba"]

text.components(separatedBy: .whitespaces)

                         → ["apple", "banana", "casaba"]
```



Extracting content from text (cont.)

- what if there are an arbitrary number of spaces between?
  - ✓ split handles it automatically; components does not

```
let spaced = "apple  banana   casaba"

spaced.split(separator: " ")

                   → ["apple", "banana", "casaba"]

spaced.components(separatedBy: " ")

                   → ["apple", "", "banana", "", "", "casaba"]

spaced.components(separatedBy: .whitespaces)

                   → ["apple", "", "banana", "", "", "casaba"]
```



Extracting content from text (cont.)

- but, can filter out empty Strings from the components array

```
let spaced = "apple  banana   casaba"

spaced.split(separator: " ")

                   → ["apple", "banana", "casaba"]

spaced.components(separatedBy: " ").filter { $0 != "" }

                   → ["apple", "banana", "casaba"]

spaced.components(separatedBy: .whitespaces).filter { $0 != "" }

                   → ["apple", "banana", "casaba"]
```

Extracting content from text (cont.)

- components also allows you to split based on newlines

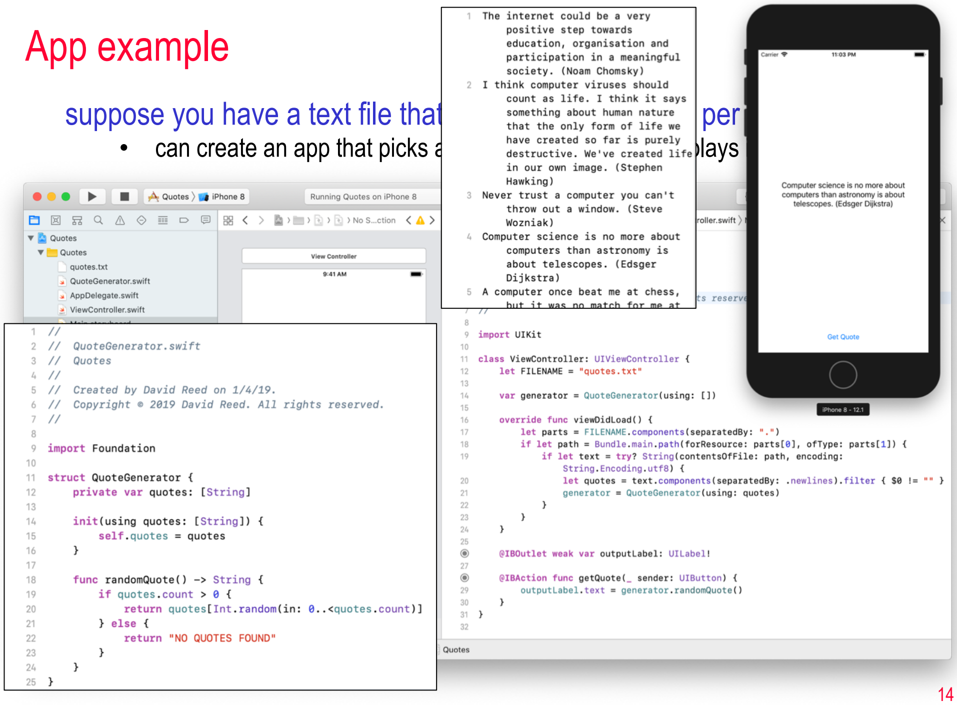
```
let multi = """"
    apple  banana
    casaba
    """


multi.components(separatedBy: .newlines)

                   → ["apple  banana", "casaba"]


multi.components(separatedBy: .whitespacesAndNewlines)

                   → ["apple", "", "banana", "casaba"]

multi.components(separatedBy: .whitespacesAndNewlines).filter
                                                       { $0 != "" }

                   → ["apple", "banana", "casaba"]
```

13

App example

suppose you have a text file that per

- can create an app that picks a plays



```
The internet could be a very
    positive step towards
    education, organisation and
    participation in a meaningful
    society. (Noam Chomsky)
I think computer viruses should
    count as life. I think it says
    something about human nature
    that the only form of life we
    have created so far is purely
    destructive. We've created life
    in our own image. (Stephen
    Hawking)
Never trust a computer you can't
    throw out a window. (Steve
    Wozniak)
Computer science is no more about
    computers than astronomy is
    about telescopes. (Edsger
    Dijkstra)
A computer once beat me at chess,
    but it was no match for me at
```

Computer science is no more about computers than astronomy is about telescopes. (Edsger Dijkstra)

```swift
//
//  QuoteGenerator.swift
//  Quotes
//
//  Created by David Reed on 1/4/19.
//  Copyright © 2019 David Reed. All rights reserved.
//

import Foundation

struct QuoteGenerator {
    private var quotes: [String]

    init(using quotes: [String]) {
        self.quotes = quotes
    }

    func randomQuote() -> String {
        if quotes.count > 0 {
            return quotes[Int.random(in: 0..<quotes.count)]
        } else {
            return "NO QUOTES FOUND"
        }
    }
}
```

```swift
//

import UIKit

class ViewController: UIViewController {
    let FILENAME = "quotes.txt"

    var generator = QuoteGenerator(using: [])

    override func viewDidLoad() {
        let parts = FILENAME.components(separatedBy: ".")
        if let path = Bundle.main.path(forResource: parts[0], ofType: parts[1]) {
            if let text = try? String(contentsOfFile: path, encoding:
                String.Encoding.utf8) {
                let quotes = text.components(separatedBy: .newlines).filter { $0 != "" }
                generator = QuoteGenerator(using: quotes)
            }
        }
    }

    @IBOutlet weak var outputLabel: UILabel!

    @IBAction func getQuote(_ sender: UIButton) {
        outputLabel.text = generator.randomQuote()
    }
}
```

14

Storing/accessing data

Bundle.main.path enables you to read from existing files

- unfortunately, it does not allow you to write to files

Swift provides a different mechanism for storing and subsequently retrieving data values

- every app has an associated Info.plist (property list) file associated with it, which stores information about the project
- it also provides storage where the app can write data & access it
- you can create and initialize multiple data storage areas if desired, but UserDefaults.standard is provided as a default storage area

```
UserDefaults.standard.set("foo", forKey: "word")
```

stores "foo" under the access key "word"

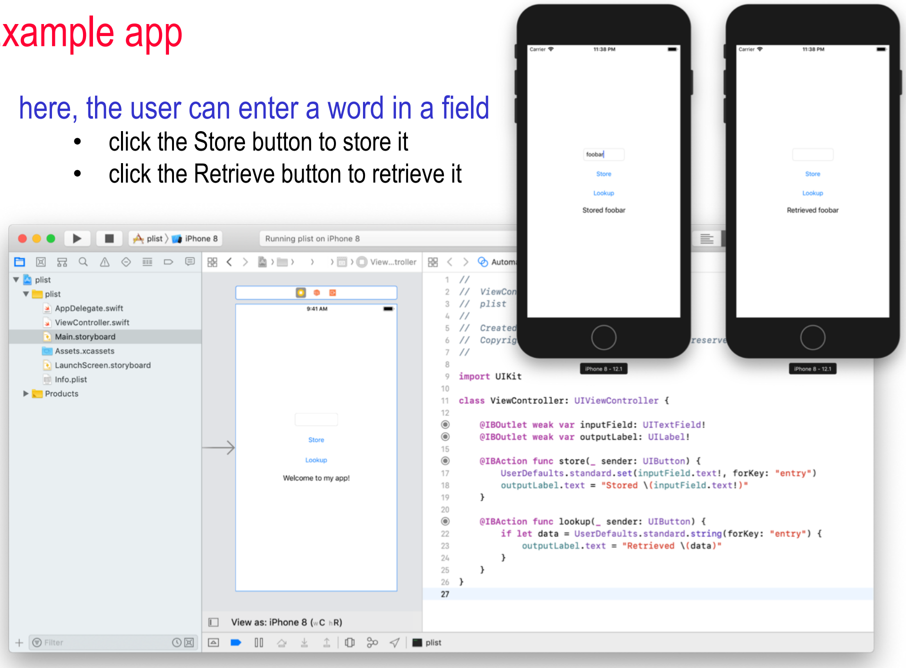
```
UserDefaults.standard.string(forKey: "word")
```

retrieves "foo" using the access key

Example app

here, the user can enter a word in a field

- click the Store button to store it
- click the Retrieve button to retrieve it



```
//
//  ViewCon
//  plist
//
//  Created
//  Copyrig                          reserve
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var inputField: UITextField!
    @IBOutlet weak var outputLabel: UILabel!

    @IBAction func store(_ sender: UIButton) {
        UserDefaults.standard.set(inputField.text!, forKey: "entry")
        outputLabel.text = "Stored \(inputField.text!)"
    }

    @IBAction func lookup(_ sender: UIButton) {
        if let data = UserDefaults.standard.string(forKey: "entry") {
            outputLabel.text = "Retrieved \(data)"
        }
    }
}
```

Storing/accessing data

can store data of any built-in type, including arrays

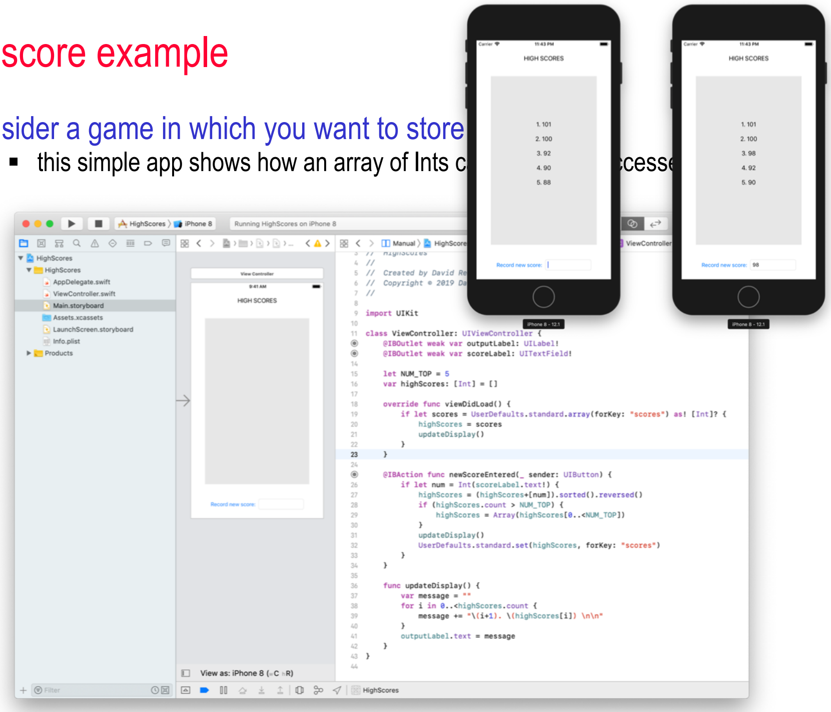
```
UserDefaults.standard.set("Chris", forKey: "name")

UserDefaults.standard.set(21, forKey: "age")

UserDefaults.standard.set(["Pat", "Alex"], forKey: "friends")

.
.
.

var who = UserDefaults.standard.string(forKey: "name")

var howOld = UserDefaults.standard.integer(forKey: "age")

var peeps = UserDefaults.standard.array(forKey: "friends") as! [String]?
```

17

High score example

consider a game in which you want to store

- this simple app shows how an array of Ints c ccesse



```
//  HighScores
//
//  Created by David Re
//  Copyright © 2019 Da
//

import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var outputLabel: UILabel!
    @IBOutlet weak var scoreLabel: UITextField!

    let NUM_TOP = 5
    var highScores: [Int] = []

    override func viewDidLoad() {
        if let scores = UserDefaults.standard.array(forKey: "scores") as! [Int]? {
            highScores = scores
            updateDisplay()
        }
    }

    @IBAction func newScoreEntered(_ sender: UIButton) {
        if let num = Int(scoreLabel.text!) {
            highScores = (highScores+[num]).sorted().reversed()
            if (highScores.count > NUM_TOP) {
                highScores = Array(highScores[0..<NUM_TOP])
            }
            updateDisplay()
            UserDefaults.standard.set(highScores, forKey: "scores")
        }
    }

    func updateDisplay() {
        var message = ""
        for i in 0..<highScores.count {
            message += "\(i+1). \(highScores[i]) \n\n"
        }
        outputLabel.text = message
    }
}
```

18

OTHER FEATURES

YOU MAY (OR MAY NOT)

FIND USEFUL

PickerView

note that PickerView is a view element, not a control element (like UIButton)

- more complex to integrate into the screen

1. add a PickerView to the scene
   - ✓ drag a PickerView from the Object Library into Interface Builder
   - ✓ position and size as desired

2. connect the PickerView to the ViewController
   - ✓ control-click from the PickerView to the View Controller icon ○ and select Outlet: dataSource
   - ✓ repeat the process and select delegate
   - ✓ add UIPickerViewDataSource, UIPickerViewDelegate

3. add the protocol names to the ViewController class

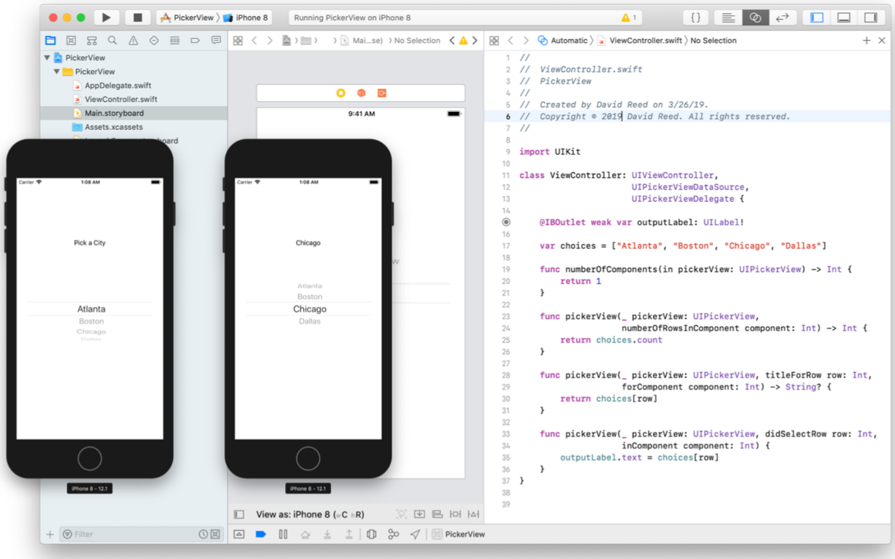
```
class ViewController: UIViewController,
                      UIPickerViewDataSource,
                      UIPickerViewDelegate {
```

PickerView (cont.)

4. finally, add methods to complete the protocol implementation

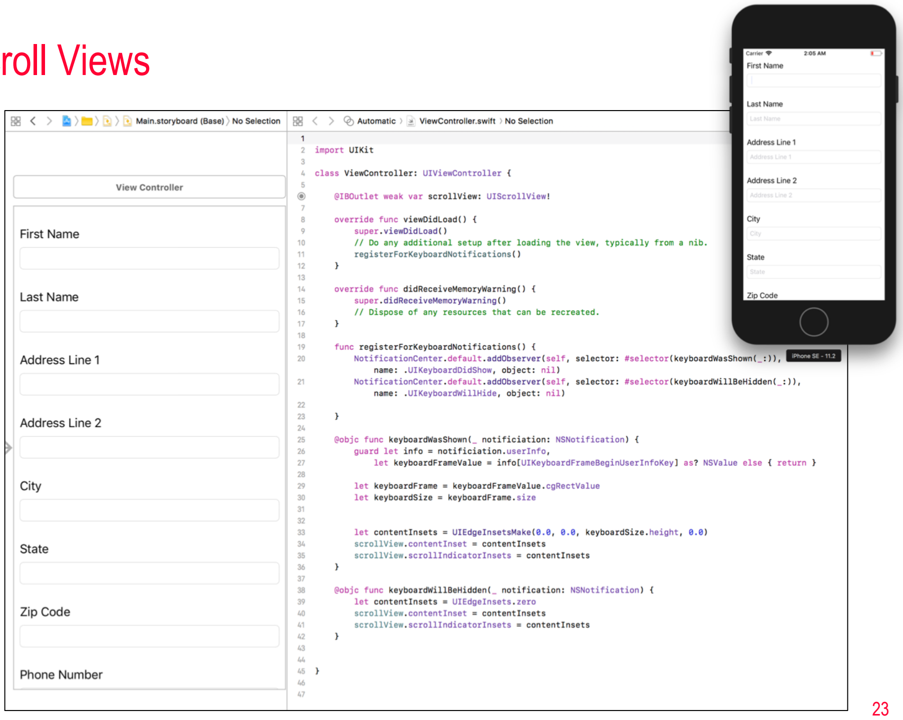
```
func numberOfComponents(in pickerView: UIPickerView) -> Int {
    // returns # of columns (usually 1)
}

func pickerView(_ pickerView: UIPickerView,
                numberOfRowsInComponent component: Int) -> Int {
    // returns the number of options to select from
}

func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
                forComponent component: Int) -> String? {
    // returns String representation of entry at the selected row
}

func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
                inComponent component: Int) {
    // specifies action to be taken when user selects a row
}
```

Example: picking cities



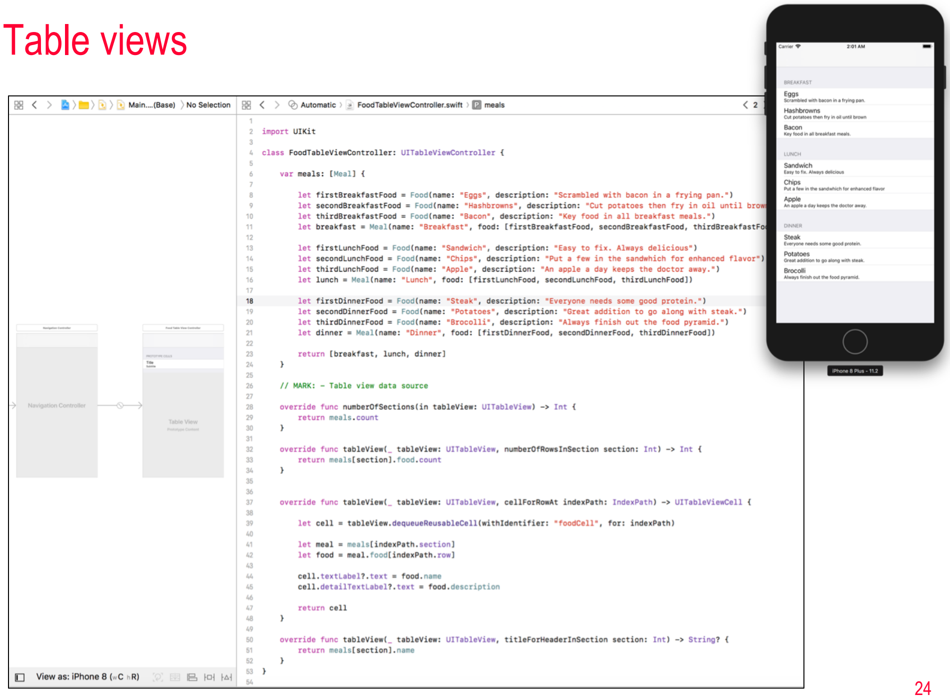
```
//
//  ViewController.swift
//  PickerView
//
//  Created by David Reed on 3/26/19.
//  Copyright © 2019 David Reed. All rights reserved.
//

import UIKit

class ViewController: UIViewController,
                      UIPickerViewDataSource,
                      UIPickerViewDelegate {

    @IBOutlet weak var outputLabel: UILabel!

    var choices = ["Atlanta", "Boston", "Chicago", "Dallas"]

    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(_ pickerView: UIPickerView,
                    numberOfRowsInComponent component: Int) -> Int {
        return choices.count
    }

    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
                    forComponent component: Int) -> String? {
        return choices[row]
    }

    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
                    inComponent component: Int) {
        outputLabel.text = choices[row]
    }
}
```

Scroll Views



```

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var scrollView: UIScrollView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        registerForKeyboardNotifications()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func registerForKeyboardNotifications() {
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWasShown(_:)),
            name: .UIKeyboardDidShow, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillBeHidden(_:)),
            name: .UIKeyboardWillHide, object: nil)

    }

    @objc func keyboardWasShown(_ notificiation: NSNotification) {
        guard let info = notificiation.userInfo,
            let keyboardFrameValue = info[UIKeyboardFrameBeginUserInfoKey] as? NSValue else { return }

        let keyboardFrame = keyboardFrameValue.cgRectValue
        let keyboardSize = keyboardFrame.size


        let contentInsets = UIEdgeInsetsMake(0.0, 0.0, keyboardSize.height, 0.0)
        scrollView.contentInset = contentInsets
        scrollView.scrollIndicatorInsets = contentInsets
    }

    @objc func keyboardWillBeHidden(_ notification: NSNotification) {
        let contentInsets = UIEdgeInsets.zero
        scrollView.contentInset = contentInsets
        scrollView.scrollIndicatorInsets = contentInsets
    }


}


```

Table views


Main....(Base)
No Selection
Navigation Controller
Food Table View Controller
PROTOTYPE CELLS
Title
Subtitle
Navigation Controller
Table View
Prototype Content
View as: iPhone 8 (wC hR)
Automatic
FoodTableViewController.swift
meals
2
Carrier
2:01 AM
BREAKFAST
Eggs
Scrambled with bacon in a frying pan.
Hashbrowns
Cut potatoes then fry in oil until brown
Bacon
Key food in all breakfast meals.
LUNCH
Sandwich
Easy to fix. Always delicious
Chips
Put a few in the sandwhich for enhanced flavor
Apple
An apple a day keeps the doctor away.
DINNER
Steak
Everyone needs some good protein.
Potatoes
Great addition to go along with steak.
Brocolli
Always finish out the food pyramid.
iPhone 8 Plus - 11.2

```

import UIKit

class FoodTableViewController: UITableViewController {

    var meals: [Meal] {

        let firstBreakfastFood = Food(name: "Eggs", description: "Scrambled with bacon in a frying pan.")
        let secondBreakfastFood = Food(name: "Hashbrowns", description: "Cut potatoes then fry in oil until brow
        let thirdBreakfastFood = Food(name: "Bacon", description: "Key food in all breakfast meals.")
        let breakfast = Meal(name: "Breakfast", food: [firstBreakfastFood, secondBreakfastFood, thirdBreakfastFo

        let firstLunchFood = Food(name: "Sandwich", description: "Easy to fix. Always delicious")
        let secondLunchFood = Food(name: "Chips", description: "Put a few in the sandwhich for enhanced flavor")
        let thirdLunchFood = Food(name: "Apple", description: "An apple a day keeps the doctor away.")
        let lunch = Meal(name: "Lunch", food: [firstLunchFood, secondLunchFood, thirdLunchFood])

        let firstDinnerFood = Food(name: "Steak", description: "Everyone needs some good protein.")
        let secondDinnerFood = Food(name: "Potatoes", description: "Great addition to go along with steak.")
        let thirdDinnerFood = Food(name: "Brocolli", description: "Always finish out the food pyramid.")
        let dinner = Meal(name: "Dinner", food: [firstDinnerFood, secondDinnerFood, thirdDinnerFood])

        return [breakfast, lunch, dinner]
    }

    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return meals.count
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return meals[section].food.count
    }


    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "foodCell", for: indexPath)

        let meal = meals[indexPath.section]
        let food = meal.food[indexPath.row]

        cell.textLabel?.text = food.name
        cell.detailTextLabel?.text = food.description

        return cell
    }

    override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
        return meals[section].name
    }
}

```

Protocols

a *protocol* defines the properties or methods that an object must have in order to complete a task

- corresponds to a Java interface; used to define the behavior of a family of classes
- e.g., CustomStringConvertible
  built-in protocol, specifies a computed property/field named `description`

```
protocol CustomStringConvertible {
    var description: String { get }
}
```

- if a class/struct implements the CustomStringConvertible protocol
  ✓ can use the computed property/field to convert to a string
  ✓ will automatically be called when you print an object
  (similar to `toString` in Java)

CustomStringConvertible

e.g., a Name struct that can be printed

```
struct Name: CustomStringConvertible {
    var first: String
    var middle: Character
    var last: String

    var description: String {
        return "\(self.first) \(self.middle). \(self.last)"
    }
}
```

```
let me = Name(first: "Dave", middle: "W", last: "Reed")
```

```
me.description              → "Dave W. Reed"
```

```
print(me)
```

Object equality

by default, user-defined objects cannot be tested for equality

```
var me = Name(first: "Dave", middle: "W", last: "Reed")
var notMe = Name(first: "Dale", middle: "F", last: "Reed")

if me == notMe {                    ➔ ERROR
    . . .
}
```

the Equatable protocol specifies a static == operator for comparisons

- operators are implemented as static methods
- defined like a method, but called like an operator: `me == notMe`

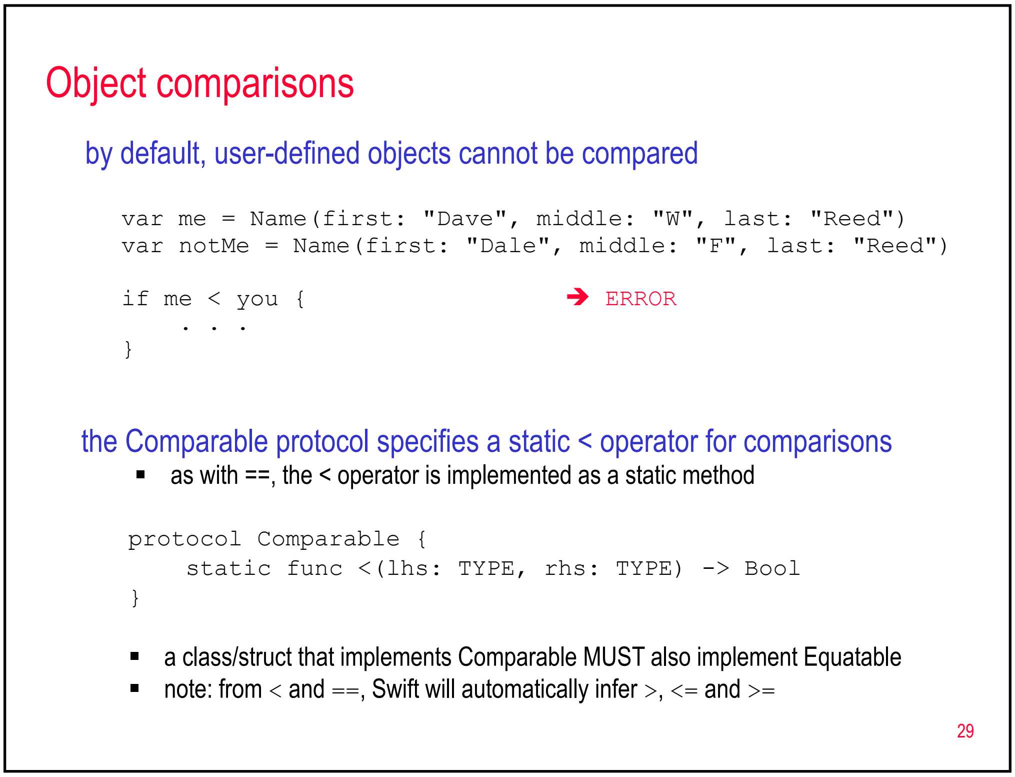
```
protocol Equatable {
    static func ==(lhs: TYPE, rhs: TYPE) -> Bool
}
```

- note: when you define ==, Swift will automatically infer `!=`

Equatable

e.g., Names with == and !=

```
class Name: CustomStringConvertible, Equatable {
    . . .

    static func ==(lhs: Name, rhs: Name) -> Bool {
        return lhs.first == rhs.first &&
            lhs.middle == rhs.middle &&
            lhs.last == rhs.last
    }
}

var me = Name(first: "Dave", middle: "W", last: "Reed")
var notMe = Name(first: "Dale", middle: "F", last: "Reed")

if me == notMe {
    print("\(me) is the same as \(notMe)")
}

if me != notMe {
    print("\(me) is different from \(notMe)")
}
```



Object comparisons

by default, user-defined objects cannot be compared

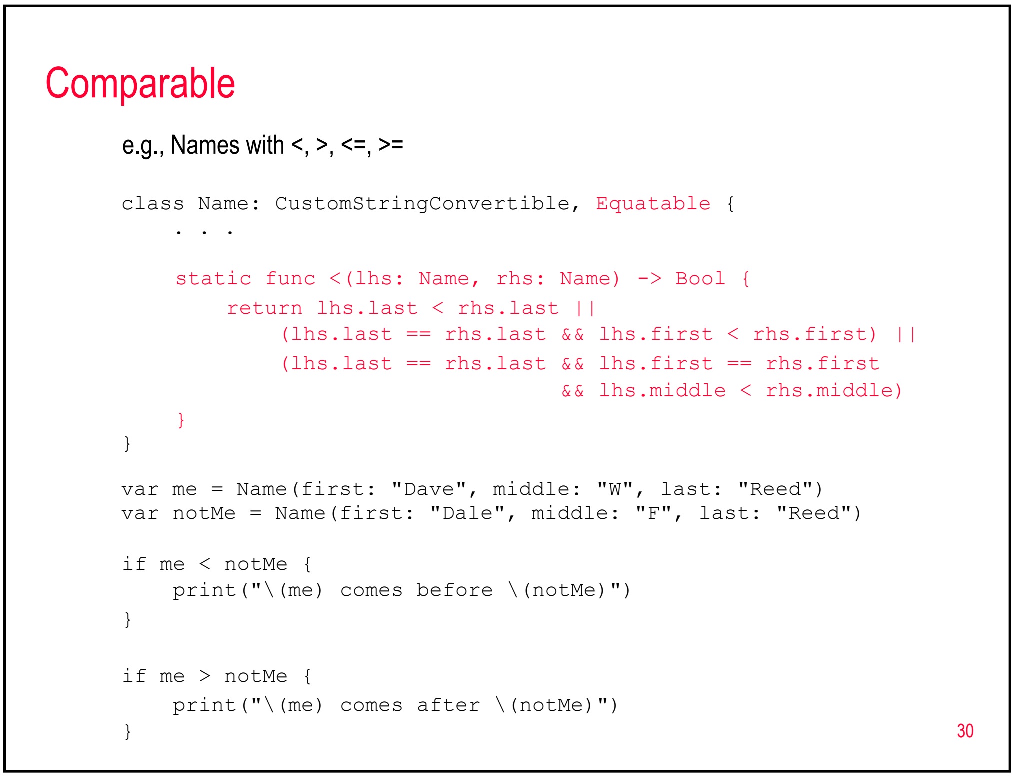
```
var me = Name(first: "Dave", middle: "W", last: "Reed")
var notMe = Name(first: "Dale", middle: "F", last: "Reed")

if me < you {                     ➔ ERROR
    . . .
}
```

the Comparable protocol specifies a static < operator for comparisons

- as with ==, the < operator is implemented as a static method

```
protocol Comparable {
    static func <(lhs: TYPE, rhs: TYPE) -> Bool
}
```

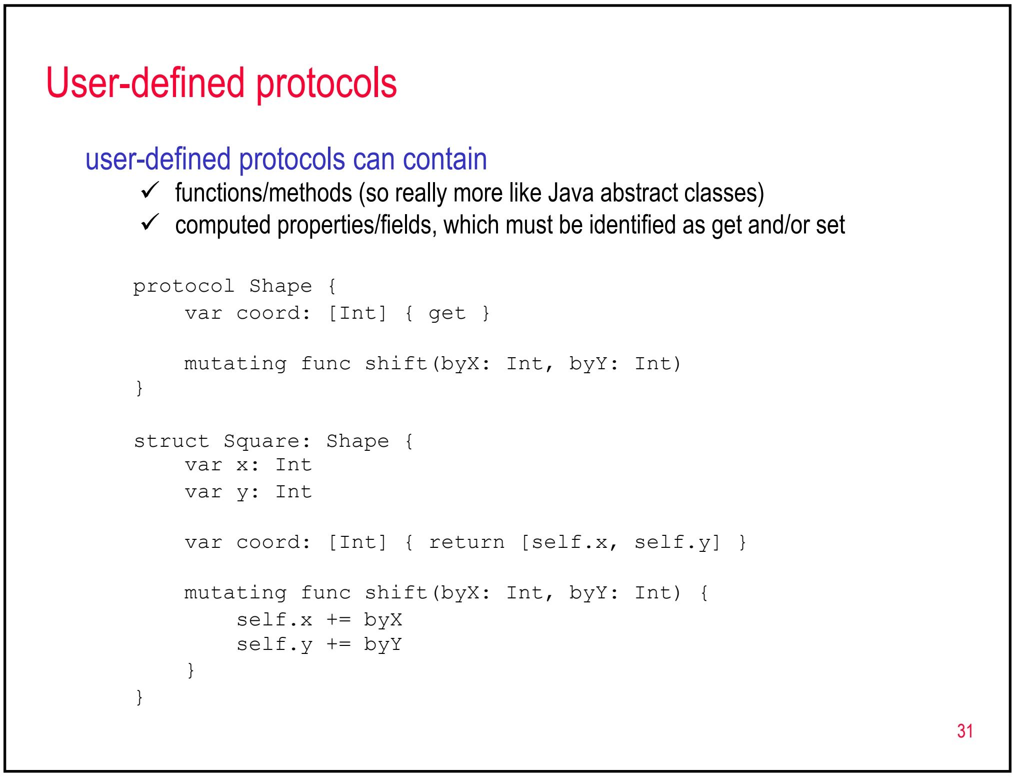
- a class/struct that implements Comparable MUST also implement Equatable
- note: from < and ==, Swift will automatically infer >, <= and >=



Comparable

e.g., Names with <, >, <=, >=

```
class Name: CustomStringConvertible, Equatable {
    . . .

    static func <(lhs: Name, rhs: Name) -> Bool {
        return lhs.last < rhs.last ||
            (lhs.last == rhs.last && lhs.first < rhs.first) ||
            (lhs.last == rhs.last && lhs.first == rhs.first
                                  && lhs.middle < rhs.middle)
    }
}

var me = Name(first: "Dave", middle: "W", last: "Reed")
var notMe = Name(first: "Dale", middle: "F", last: "Reed")

if me < notMe {
    print("\(me) comes before \(notMe)")
}

if me > notMe {
    print("\(me) comes after \(notMe)")
}
```

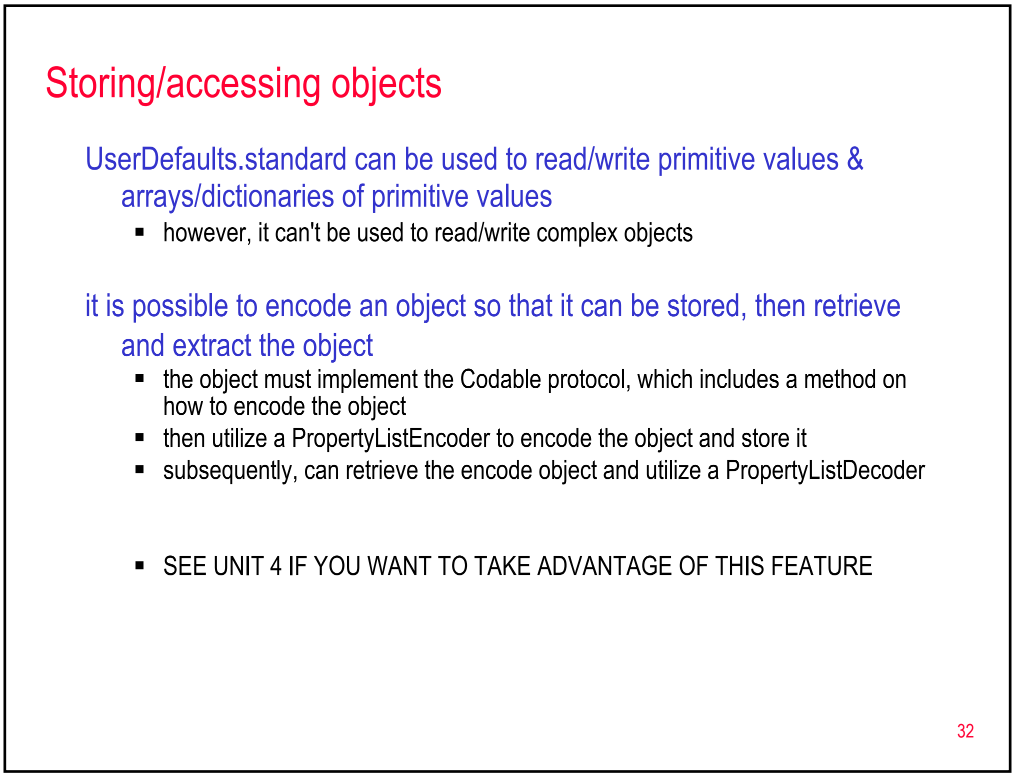


User-defined protocols

user-defined protocols can contain

- ✓ functions/methods (so really more like Java abstract classes)
- ✓ computed properties/fields, which must be identified as get and/or set

```
protocol Shape {
    var coord: [Int] { get }

    mutating func shift(byX: Int, byY: Int)
}

struct Square: Shape {
    var x: Int
    var y: Int

    var coord: [Int] { return [self.x, self.y] }

    mutating func shift(byX: Int, byY: Int) {
        self.x += byX
        self.y += byY
    }
}
```



Storing/accessing objects

UserDefaults.standard can be used to read/write primitive values & arrays/dictionaries of primitive values

- however, it can't be used to read/write complex objects

it is possible to encode an object so that it can be stored, then retrieve and extract the object

- the object must implement the Codable protocol, which includes a method on how to encode the object
- then utilize a PropertyListEncoder to encode the object and store it
- subsequently, can retrieve the encode object and utilize a PropertyListDecoder

- SEE UNIT 4 IF YOU WANT TO TAKE ADVANTAGE OF THIS FEATURE