CSC 581: Mobile App Development

Spring 2018

Unit 4: Swift protocols & design

- protocols
  - CustomStringConvertible
  - Equatable, Comparable
  - user-defined protocols
- MVC pattern
  - Hunt the Wumpus example
- useful techniques
  - PickerView
  - Popover windows
  - App icons

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 computing property/field named `description`

```
class Name: CustomStringConvertible {
    private var first: String
    private var middle: Character
    private var last: String

    init(first: String, middle: Character, last:String) {
        self.first = first
        self.middle = middle
        self.last = last
    }

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

CustomStringConvertible

- if a class/struct implements the CustomStringConvertible protocol
  - ✓ can use the computer property/field to convert to a string
  - ✓ will automatically be called when you print an object

```
class Name: CustomStringConvertible {
    private var first: String
    private var middle: Character
    private var last: String

    . . .

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


var me = Name(first: "David", middle: "W", last: "Reed")

me.description      → "David 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 == you {      → ERROR
    . . .
}
```

the Equatable protocol specifies a static == operator for comparisons

- operators are implemented as static methods
- you don't call like a normal method: `me.==(you)` instead, `me == you`

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

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

Equatable

- if a class/struct implements the Equatable protocol
  - ✓ can test objects for equality using == operator
  - ✓ can also test for inequality using != (its definition is inferred from ==)

```
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 {
  // TEST EVALUATES TO false
}

if me != notme {
  // TEST EVALUATES TO true
}
```

5

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

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

 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)
     }
}
```

6

Comparable

- a class/struct that implements Comparable must also implement Equatable
  - ✓ can compare objects using <; but also >, <=, and >=
  - ✓ the other operators are inferred from < and ==

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

    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 {
  // TEST EVALUATES TO false
}

if me >= notme {
  // TEST EVALUATES TO true
}
```

7

User-defined protocols

user-defined protocols can contain

- ✓ functions/methods
- ✓ computed properties/fields (must be identified as get and/or set)

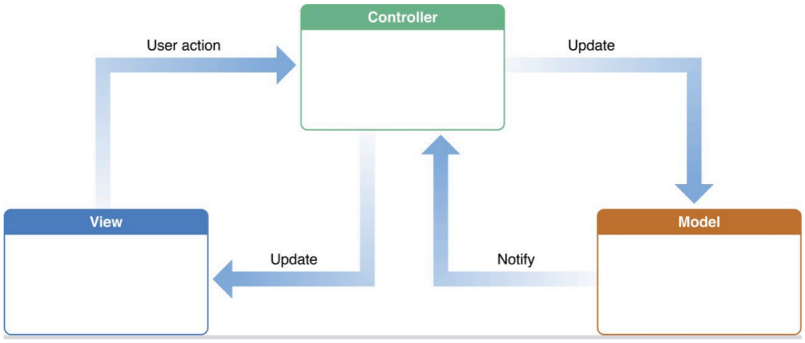
```
protocol Shape {
    var coord: [Int] { get }

    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
    }
}
```

8

MVC revisited

recall: the Model-View-Controller pattern is commonly used in software engineering to build modular, reusable code

- the model defines the logic (i.e., user-defined Swift files)
- the view is the interface the user interacts with (e.g., screens, buttons, images, …)
- controllers are code/utilities that connect the model & view (e.g., UIViewController)

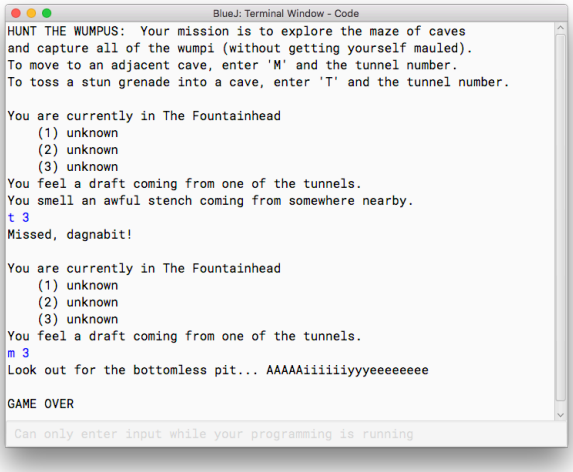


9

Design example: Hunt the Wumpus

recall from a CSC 222 assignment:

- Hunt the Wumpus was a text-based video game from the 70's

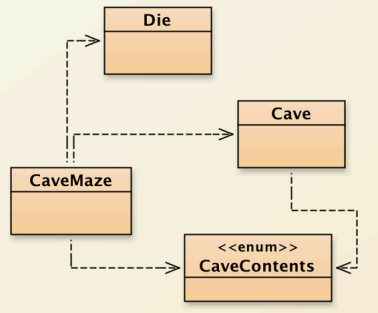


- player explores a maze of caves, with each cave connected to 1-4 others
- randomly placed wumpi (1-3), bottomless pit (1) and bat swarm (1)
- player can sense when an obstacle is adjacent
- player can move or throw a stun grenade through a tunnel, wumpi move when hear an explosion

goal: avoid obstacles and capture all of the wumpi before they maul you!

10

CSC 222: Hunt the Wumpus design



Cave:

- contains all of the information about a given cave, including its contents

CaveContents:

- enumeration for specifying the possible contents of a cave

CaveMaze:

- models the maze of caves
- the caves stored in a list, linked together
- utilizes the Die class in order to select random locations in the maze

WumpusTerminal:

WumpusGUI:

- drivers for playing the game

11

CaveContents (in Java)

```
public enum CaveContents {
    EMPTY, WUMPUS, BATS, PIT
}
```

in Swift?

12

Cave (in Java)

```
public class Cave {
    private String name;
    private int num;
    private ArrayList<Integer> adjacent;
    private CaveContents contents;
    private boolean visited;

    public Cave(String name, int num, ArrayList<Integer> adj) {
        // INITIALIZES ALL OF THE FIELDS
    }

    public String getCaveName() {
        // RETURNS THE CAVE NAME IF VISITED, ELSE "unknown"
    }

    public int getCaveNumber() {
        // RETURNS THE CAVE NUMBER
    }

    public int getNumAdjacent() {
        // RETURNS THE NUMBER OF ADJACENT CAVES
    }

    public int getAdjNumber(int tunnel) {
        // RETURNS THE NUMBER OF THE CAVE CONNECTED BYTHE SPECIFIED TUNNEL
    }

    public void setContents(CaveContents contents) {
        // SETS THE CONTENTS OF THE CAVE TO THE SPECIFIED VALUE
    }

    public CaveContents getContents() {
        // RETURNS THE CONTENTS OF THE CAVE
    }

    public void markAsVisited() {
        // MARKS THE CAVE AS VISITED (SO IT IS NO LONGER UNKNOWN)
    }
}
```

in Swift?

13

CaveMaze

```
public class CaveMaze {
  private int numGrenades, numWumpi;
  private Cave currentCave;
  private boolean able;
  private ArrayList<Cave> caves;
  private Die d;

  public CaveMaze(String filename) throws java.io.FileNotFoundException {
    // READS IN CAVE DATA AND PLACES WUMPI, BATS & PIT IN RANDOM CAVES
  }

  public String move(int tunnel) {
    // MOVES THE PLAYER, CHECKS FOR OBSTACLES, RETURNS RESULT
  }

  public String toss(int tunnel) {
    // TOSSES A GRENADE & POSSIBLY MOVES WUMPI, RETURNS RESULT
  }

  public String showLocation() {
    // RETURNS CURRENT LOCATION AND ANY WARNINGS
  }

  public boolean stillAble() {
    // RETURNS TRUE IF PLAYER NOT DISABLED
  }

  public boolean stillWumpi() {
    // RETURNS TRUE IF ANY WUMPI REMAINING
  }
}
```

for HW4, a Swift implementation of CaveMaze is provided

14

HW4: Hunt the Wumpus app

you will design and build an app for playing Hunt the Wumpus

- it MUST use the model provided by CaveContents, Cave, and CaveMaze
- it should have an opening screen with some graphic element, author name, and a button for beginning the game
- it should have a play screen with a label for displaying the location and results of each action, and UI elements that enable the player to move or toss down a specific tunnel
- it should have a help button which brings up a popover window with instructions
- it should have a restart button for beginning a new game
- it should have an app icon of your own design

15

Hunt the Wumpus design

for Hunt the Wumpus,

- want to think carefully about how to specify actions
- note that the number of tunnels varies by cave, so your interface must adjust
- option 1: a MOVE button for each tunnel, a TOSS button for each tunnel
- option 2: singe MOVE and TOSS buttons, with a text field for entering the tunnel #
- option 3: use a PickerView

- option 4: ???

16

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 {
```

17

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
}
```

18

Example: picking cities

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

    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]
    }

    @IBOutlet weak var outputLabel: UILabel!

}
```

19

Example: picking numbers

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

    var low = 4
    var high = 8

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

    func pickerView(_ pickerView: UIPickerView,
                    numberOfRowsInComponent component: Int) -> Int {
        return high-low+1
    }

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

    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
                    inComponent component: Int) {
        outputLabel.text = Strig(row+low)
    }

    @IBOutlet weak var outputLabel: UILabel!

}
```

20

Popovers

also for Hunt the Wumpus

- want to be able to get help/instructions at any time
- could do this by having separate views & segue
- more attractive solution – a popover window

to open a popover (at the click of a button)

1. add a popover View to the storyboard
   - ✓ drag a View from the Object Library onto the bar at the top of the ViewController

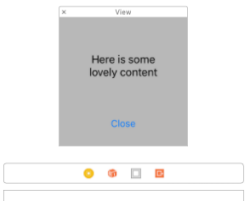
   - ✓ this will add a View icon to the bar and also display the View in the storyboard

21

Popovers (cont.)

2. specify the background & size for the popover
   - ✓ select the View, go to the Attributes Inspector and set background color
   - ✓ select the View, go to the Size Inspector and set X & Y

3. add content to the popover View
   - ✓ e.g., a label for displaying text and a button for closing



4. connect the popover View to the ViewController
   - ✓ i.e., control-drag from the View to create an outlet in the ViewController code (be careful to connect the UIView, not the contained elements)

```
@IBOutlet var popoverView: UIView!
```

22

Popovers (cont.)

5. create an Action for the button in the ViewController (to open the popover)
   - ✓ add Swift statements that open & position the popover

```
@IBAction func openPopover(_ sender: UIButton) {
    self.view.addSubview(popoverView)
    popoverView.center = self.view.center
}
```

6. create an Action for the button in the popover View (to close the popover)
   - ✓ add a Swift statement to close the popover

```
@IBAction func closePopover(_ sender: UIButton) {
    self.popoverView.removeFromSuperview()
}
```

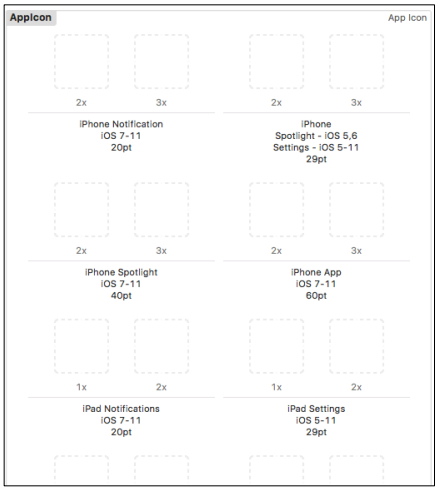
7. if desired, you can round the edges of the popover
   - ✓ select the View and go to the Identify Inspector
   - ✓ create a new User Defined Runtime Attribute (by clicking on +)
   - ✓ Key Path = `layer.cornerRadius`, Type = `Number`, Value = `10`

23

App icons

attaching an icon to your app is simple

- go to the Assets.xcassets folder, and click on AppIcon
- this will open a collection of frames for uploading your icon images
- to accommodate different devices, need to supply many files of different sizes
- there are many online sites that will convert an image into the full range of sizes
  e.g., http://appiconmaker.co
- for best results, start with an image that is square and fairly simple



24