IS 543 Fall 2023

Mobile Platform Development

More Enum and Optional

Error Handling

Playing Sounds

Dr. Stephen W. Liddle
782 TNRB, office hours Tu 9:30-10:30am, Th 2-3pm
is543@byu.edu
801-422-8792 (rings on my cell)
Zoom: https://bit.ly/liddlezoom

Today

- We'll cover a few more `enum` and `Optional` ideas
- Then error handling: `throw`, `throws`, `try`, `do-catch`
- And how to use AVFoundation to play sounds

Fun Facts

- How many grandchildren does Professor Liddle have?
  A) None
  B) 4
  C) 7
  D) 10
  E) None of the above

6 children + 6 spouses + 10 grandchildren + 1 granddaughter who is imminent

Children's children are the crown of old men;
and the glory of children are their fathers.

Proverbs 17:6

HW5 Debrief

- How good are the LLM-based generative AI tools?
- What did you like/dislike about interacting with them?
- What did you learn about functional programming?
- What did you learn about singletons?
- How helpful were the videos? What did you like/dislike about them?

enum

- Getting all the cases of an enumeration (adopt `CaseIterable` protocol)

```
enum TeslaModel: CaseIterable {
    case X
    case S
    case Three
    case Y
}
```

Now this `enum` will have a `static var allCases` that you can iterate over

```
for model in TeslaModel.allCases {
    reportSalesNumbers(for: model)
}

func reportSalesNumbers(for model: TeslaModel) {
    switch model { ... }
}
```

Optionals

- Optional

  Declaring something of type `Optional<T>` can be done with the syntax `T?`

```
enum Optional<T> {
    case none
    case some(<T>)
}
```

```
var hello: String?
var hello: String? = "hello"
var hello: String? = nil
```

```
var hello: Optional<String> = .none
var hello: Optional<String> = .some("hello")
var hello: Optional<String> = .none
```

Optionals

- Optional

  Declaring something of type `Optional<T>` can be done with the syntax `T?`

  You can then assign it the value `nil` (i.e. `Optional.none`)

```
enum Optional<T> {
    case none
    case some(<T>)
}
```

```
var hello: String?
var hello: String? = "hello"
var hello: String? = nil
```

```
var hello: Optional<String> = .none
var hello: Optional<String> = .some("hello")
var hello: Optional<String> = .none
```

Optionals

- Optional

  Declaring something of type `Optional<T>` can be done with the syntax `T?`

  You can then assign it the value `nil` (i.e. `Optional.none`)

  Or you can assign it something of type `T` (`Optional.some` with associated data value of type `T`)

```
enum Optional<T> {
    case none
    case some(<T>)
}
```

```
var hello: String?
var hello: String? = "hello"
var hello: String? = nil
```

```
var hello: Optional<String> = .none
var hello: Optional<String> = .some("hello")
var hello: Optional<String> = .none
```

Optionals

- Optional

  Declaring something of type `Optional<T>` can be done with the syntax `T?`

  You can then assign it the value `nil` (i.e. `Optional.none`)

  Or you can assign it something of type `T` (`Optional.some` with associated data value of type `T`)

  Note that `Optional` vars always start out with an implicit `= nil` assignment

```
enum Optional<T> {
    case none
    case some(<T>)
}
```

```
var hello: String?
var hello: String? = "hello"
var hello: String? = nil
```

```
var hello: Optional<String> = .none
var hello: Optional<String> = .some("hello")
var hello: Optional<String> = .none
```

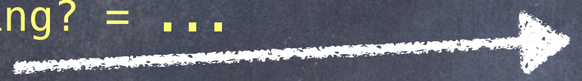
Optionals

- Optional

  You can access the associated value either by "force-unwrapping" (with !) ...

```
enum Optional<T> {
    case none
    case some(<T>)
}

let hello: String? = ...
print(hello!)
```



```
switch hello {
    case .none:  // raise an exception (i.e. crash)
    case .some(let data): print(data)
}
```

And I strongly recommend that you not do this!

Optionals

- Optional

  You can access the associated value either by "force-unwrapping" (with !) ...

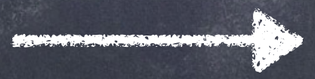
  Or "safely" using if let and then using the safely-retrieved associated value in { } (else allowed too)

```
enum Optional<T> {
    case none
    case some(<T>)
}
```

```
let hello: String? = ...
print(hello!)
```

```
switch hello {
    case .none:  // raise an exception (i.e. crash)
    case .some(let data): print(data)
}
```

```
if let safeHello = hello {
    print(safeHello)
} else {
    // do something else
}
```



```
switch hello {
    case .none:  // do something else
    case .some(let safeHello): print(safeHello)
}
```

Optionals

- Optional

  You can access the associated value either by "force-unwrapping" (with `!`) …

  Or "safely" using `guard let … else` and then using the safely-retrieved value after the `{ }`

```
let hello: String? = ...

guard let safeHello = hello else {
    // you must exit the scope, e.g. return, throw, or fatalError()
}

// here, safeHello is unwrapped
print(safeHello)
```

```
switch hello {
    case .none:  // exit the scope
    case .some(let safeHello): print(safeHello)
}
```

- We may use `guard-let-else` rather than nesting `if-let` statements

  And sometimes it just feels more natural

Optionals

- Optional

  The "nil-coalescing operator" ?? gives an Optional expression a default value

```
enum Optional<T> {
    case none
    case some(<T>)
}

let x: String? = ...

let y = x ?? "some default"
```

```
switch x {
    case .none: y = "some default"
    case .some(let data): y = data
}
```

Optional Chaining

- Optional chaining lets you safely access paths that may be nil

  The ?. operator returns nil and stops evaluating the expression if the left-hand side is nil

```
class Person {
    var residence: Residence?
}

class Residence {
    var numberOfRooms = 1
}

let john = Person()
john.residence = Residence()

if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
```

Optional Chaining

- You can build access chains too:

```
class Person {
    var residence: Residence?
}

class City {
    var name = "Provo"
}

let john = Person()
john.residence = Residence()
john.residence?.city = City()
john.residence?.city?.name = "Orem"

if let cityName = john.residence?.city?.name {
    print("John's city of residence is \(cityName).")
} else {
    print("Unable to retrieve the city name.")
}
```

```
class Residence {
    var numberOfRooms = 1
    var city: City?
}
```

Code still works when something is nil

throw

- In Swift you can `throw` error objects

  I.e. any type that conforms to the `Error` protocol

```
enum VendingMachineError: Error {
    case InvalidSelection
    case InsufficientFunds(coinsNeeded: Int)
    case OutOfStock
}

throw VendingMachineError.InsufficientFunds(coinsNeeded: 5)
```

Handling Errors 1/4: throws

- You can choose to propagate the error

```
func myHelperMethod() throws {
    // If any of your code throws an error object, that gets propagated to the
    // code that called myHelperMethod()
}

func myHelperMethod() throws -> String {
    // Keyword "throws" goes before the arrow if the function has a return type
}
```

Handling Errors 2/4: do/catch

- You can catch the error

  Surround error-throwing code with do { } block

  Attach catch { } blocks as needed

```
do {
    try expression
    statements
} catch pattern1 {
    statements
} catch pattern2 where condition {
    statements
}
```

```
var vendingMachine = VendingMachine()

vendingMachine.coinsDeposited = 8

do {
    try buyFavoriteSnack("Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.InvalidSelection {
    print("Invalid Selection.")
} catch VendingMachineError.OutOfStock {
    print("Out of Stock.")
} catch VendingMachineError.InsufficientFunds(let coinsNeeded) {
    print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")
} catch {
    print("Unexpected error.")
}
```

Handling Errors 3/4: try?

- Convert the error to an optional value: try?

```
func fetchDataFromDisk() throws -> Data {
    // ...
}

func fetchDataFromServer() throws -> Data {
    // ...
}

func fetchData() -> Data? {
    if let data = try? fetchDataFromDisk()   { return data }
    if let data = try? fetchDataFromServer() { return data }
    return nil
}
```

Handling Errors 4/4: try!

- Disable error propagation: try!

```
let photo = try! loadImage("./Resources/John Appleseed.jpg")
```

- For example, with AVFoundation:

```
// Assert that AVAudioPlayer, which might throw an error, never really does
audioPlayer = try! AVAudioPlayer(contentsOf: wrongSound)
```

- So remember the three variants of try:

try "Normal" try: throw error on exception
try! "Override" try: assert that the code will not throw an error (runtime error if it does)
try? "Optional" try: return nil on error rather than throwing an error

Optionals Revisited

- Swift relies a LOT on the concept of `Optional<Type>`

  This is great for us, but it also can cause some headaches:

```
if let path = Bundle.main.path(forResource: "sound.mp3", ofType:nil) {
    if let ... {
        if let ... {
            // The nesting can keep going...
        }
    }
}
```

- You can also use a `guard` variation:

```
guard let path = Bundle.main.path(forResource: "sound.mp3", ofType:nil) else {
    return    // This code block must exit (i.e. usually return from function)
}

// From here forward, path is guaranteed to be non-nil, so it's not Optional
```

Optionals Revisited

- Handy operators for Optionals:
  Nil-coalescing "??" (a.k.a. the "optional default" operator)

```
func playSound(defaultPath: Path) {
    let path = Bundle.main.path(forResource: "sound.mp3", ofType:nil)
    // path is of type Optional<Path>

    let url = URL(fileURLWithPath: path ?? defaultPath) // Use default if path is nil
    ...
}
```

- Optional chaining (another use of "?"):

```
var player: AVAudioPlayer?
...
player?.play() // If player is not nil, invoke play(), but stop executing if nil
               // An expression with optional chaining returns nil if its LHS is nil
```

Playing Sound Effects

- AVFoundation handles multimedia playback and recording (audio, video)

  Simple recipe for playing an embedded sound file:

```
struct SoundPlayer {
    var player: AVAudioPlayer?

    mutating func playSound(named soundName: String) {
        guard let path = Bundle.main.path(forResource: soundName, ofType: nil) else {
            return
        }

        do {
            player = try AVAudioPlayer(contentsOf: URL(fileURLWithPath: path))
            player?.play()
        } catch {
            // Ignore -- the sound just won't play
        }
    }
}
```

Playing Sound Effects

- AVFoundation handles multimedia playback and recording (audio, video)

  Simple recipe for playing an embedded sound file:

```
struct SoundPlayer {
    var player: AVAudioPlayer?

    mutating func playSound(named soundName: String) {
        guard let path = Bundle.main.path(forResource: soundName, ofType: nil) else {
            return
        }

        do {
            player = try AVAudioPlayer(contentsOf: URL(fileURLWithPath: path))
            player?.play()
        } catch {
            // Ignore -- the sound just won't play
        }
    }
}
```

Why not make this a local variable in playSound()?

The call to play() runs a background play operation

When playSound() completes, a local variable would be destroyed, so the player object would be deleted before it finished playing the sound

Fun with AVAudioPlayer

- You can do a LOT with the powerful AVAudioPlayer, such as:
  Play sounds (from memory buffers or from files)
  Loop the playing of sounds
  Play sound at a future time
  Synchronize multiple players working simultaneously
  Control playback level, stereo positioning, playback rate
  Seek to a point in a sound file (i.e. rewind, fast-forward)
- For example, here's a recipe for playing faster than normal:

```
// Given an AVAudioPlayer called player...
player.enableRate = true    // Turn on rate control
player.prepareToPlay()      // Load up memory buffers with relevant data
player.rate = 2.8           // Set playback rate to 2.8x normal
```

Architecture of Playing Sounds

- Where should we create a SoundPlayer?
  Model?
  View?
  ViewModel?

  Model layer is problematic, because playing sound is a view-oriented thing

  View layer seems problematic, because it gets re-rendered whenever the model changes
  But perhaps we could use some of the handlers like `.onAppear` or button action to start playing

  When deciding whether to make the View or the ViewModel more complex, favor a simple View
  "By default" we would look to the ViewModel as the layer for our sound effects
  Semantically, it feels good to associate sound effects with user Intents (i.e. in the ViewModel)

Managing the AVAudioSession

- Default iOS AVAudioSession has these characteristics
  - Supports playback but not recording (i.e. no microphone access)
  - Setting the hardware Ring/Silent switch to silent mode prevents audio from playing on the speaker
  - Locking the device also silences the app's audio
  - When your app plays sound, it silences any other background audio
- Should game sound effects mix with audio from other apps?
  - If yes, the recipe for changing the session settings is easy:

```
try? AVAudioSession.sharedInstance().setCategory(.ambient)
```

  - Remember that try? "swallows" any errors thrown and instead returns nil
  - AVAudioSession uses the singleton pattern
  - Change the shared instance's category to change the mixing characteristics
  - You can actually change a combination of category and mode (see the documentation)

Let's Record and Play Some Sounds

- Demo time!
  Let's add "click" sounds to our calculator

Homework 6 for Thursday

- Due before class on Thursday:

  Repeat what we did in class to enable your HW4 calculator UI to play sounds when buttons are tapped. Add an element to your UI that lets the user enable or disable sounds as they wish. (An easy way is to add an HStack with a Text label and a Switch.)

  Now begin to construct the Model and ViewModel layers of your calculator app. Think about what state information is captured in Apple's simple 4-function calculator app that we are emulating. How would you represent that in the Model? What are the model accessors and user intents your ViewModel should support? Begin to outline the code. You don't need to finish the calculator engine for HW6, but make some progress.

  Upload to Learning Suite a PDF with your Model, ViewModel, and View code along with a screenshot of the running UI that includes your sound on/off feature.

Today's Forum

- This will be an interesting talk!

  Dr. Akhil Reed Amar

  Sterling Professor of Law and Political Science, Yale University

  Teaches constitutional law

  Graduated from Yale College, summa cum laude, in 1980 and from Yale Law School in 1984

  Clerked for Stephen Breyer

  Joined the Yale faculty in 1985 at the age of 26

  Has won awards from both the American Bar Association and the Federalist Society

  Cited by Supreme Court justices about 50 times